White Paper
Docland XML Engine (DXe)

Garrett O'Carroll, Docland Software.
Rev. 6.0 7/6/06

Summary

Docland's **XML Component** Technology provides a new way to build dynamic web pages using Object-Oriented techniques and "XML Components". Document assembly using XML Components has many benefits , in the same way that use of software components benefits building software systems. Client -side document assembly using **AJAX**[1] functionality in a web browser also offers a way to reduce web server traffic and server workload.

Benefits:

- Accelerate web site development
- Improve consistency of web application presentation
- Improve quality & reliability by reducing project size
- Reduce network traffic by means of client-side component caching
- Reduce server load & transfer the cost of page assembly work from expensive server resources to your user's browsers
- Gain productivity by using "Object Oriented" processes for web page design
- Reduce maintenance costs & enable rapid, consistent site updates by eliminating content duplication

DXe has been implemented in two variants:

- DXe/AJAX performs client side document assembly using JavaScript and XML Components.

- DXe/J2EE performs document assembly within the J2EE server

The process implemented in this software is patented in the USA (6,772,165) and other countries.

Introduction

Docland Software has developed XML Component technology that delivers significant benefits to developers of web applications. The technology is implemented in the Docland XML engine (DXe). DXe delivers a new method for assembling web pages from reusable components of XML. This method can be applied to web application presentation layers, content caching, and content management systems. DXe enables web developers to move beyond today's monolithic approach to web page generation and to utilise powerful component based techniques for building web pages and other XML documents.

With DXe, the presentation layer of a web application can be developed as a set of reusable XML components. These components can be assembled in hierarchical fashion to generate the web pages for the application.

[1] Asynchronous JavaScript And XML

DXe complements and adds value to current presentation technologies such as JSP and XMLC, without impinging on the design of the business logic programs of a system.

DXe/AJAX implements page assembly in the web browser and can reduce web traffic by caching common components locally and avoiding transferring these page components repeatedly at each page request. Depending on the page, this can reduce web traffic dramatically. An added benefit is that complex page assembly tasks no longer have to be preformed by expensive server hardware, thus improving data centre resource utilisation.

What's the problem?

Web pages can be very large and complex. This is especially true for web pages that form the user interface of web applications. To achieve a consistent look and feel across a web site, pages often have large amounts of common content such as menus, page headers and page footers.

Duplication of content causes problems when maintaining or developing web sites. If an error is found in the duplicated content, or if a change is required, then every page in the site must be changed. Often, content, which appears to be similar on a set of pages, is in fact produced by subtly differing HTML mark-up, so it is not possible to use a common include in such cases.

Also, as applications grow more complex, the need inevitably arises to integrate information from a variety of services onto the web application's front-end. For example:

> *"If you hit the Amazon.com gateway page, the application calls more than 100 services to collect data and construct the page for you."*
> *(WERNER VOGELS Amazon CTO,*
> *"Learning from the Amazon technology platform", http://www.acmqueue.com)*

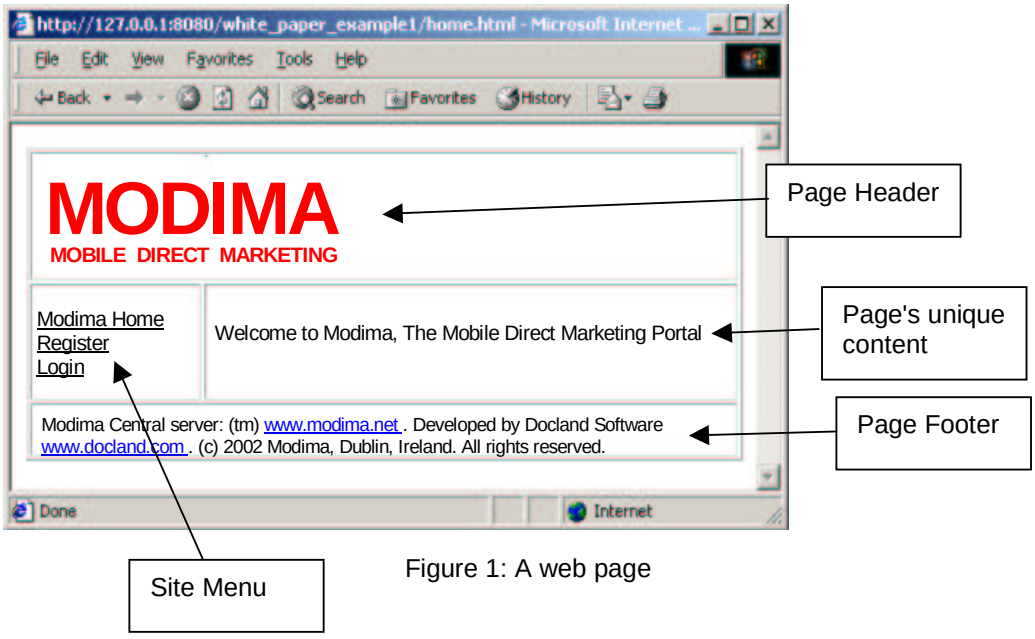

Figure 1: A web page

The "Include-File" Solution

This problem is usually solved using include-files. "Include-files" allow developers to place commonly used content in a separate file and to have the server insert this file in a particular location in a page whenever the page is sent out to a browser. This is a simple solution to the problem. Using include-files, only one file needs to be maintained for each piece of shared content.

As web pages become more complex, the limitations of include-files become apparent. The web page shown in Figure 1 contains a page header, page footer and a site menu. It also contains some information unique to that page ("Welcome to Modima" etc).

The web page shown in figure 1 can be build using 4 include-files. Figure 2 shows how this can be written.

```
<html>
        <head>
        </head>
        <body>
                <%@ include file="header1.html " %>
                <%@ include file="menu.html " %>
                <%@ include file="header2.html " %>

                Welcome to Modima, The Mobile Direct Marketing Portal

                <%@ include file="footer.html " %>
        </body>
</html>
```

Figure2 : HTML with include-file directives.

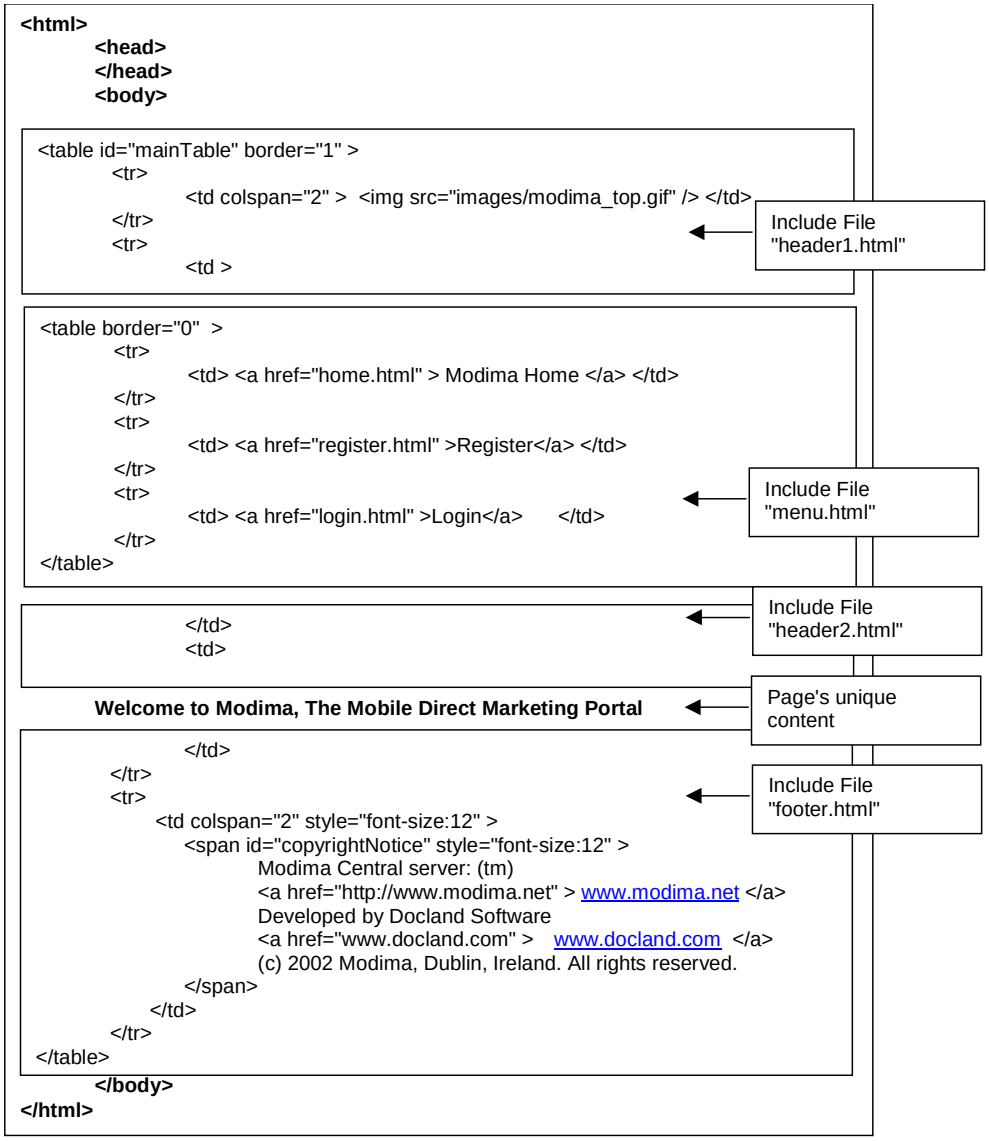

Figure 3: HTML source with include-files.

Figure 3 shows the HTML generated when the "include-file" directives are replaced with the content of the include-files.

This example shows how the include-files are related to one another. For example, the table "mainTable" spans the whole page. Its opening tag is in the file header1.html, and its closing tag is in file footer.html. There is therefore a relationship between the two include-files - header1.html and header2.html. This relationship is not explicitly stated anywhere, and it is the responsibility of the page developer to be aware of this relationship and to ensure that both files are always included in each page. This is simple example of how dependencies can arise between include-files.

Include-files have a number of innate disadvantages:

- File often contain disjointed mark-up for example: a table opening tag may be contained in one header file, and the closing tag for the same table in another include file. This complicates maintenance and development.

- Include files are inserted en-bloc without modification. They offer no mechanism for refining content by "inheriting" common content adding some more specific elements. For example, a web site's menu often varies from page to page, but all menus share common first (e.g. "Home") and last ("About") links. Entries in between may vary depending on the page. Include files offer no mechanism to express this type of relationship.

Figure 4 below shows how this page might be written using DXe & XML Components. Note how each component is a complete and well-formed XML document. The dxe:extends attribute indicates the component from which this page inherits. It is analogous to the "extends" keyword in the Java Object Oriented Programming language. A further example is given later in this document.

```
<html dxe:extends="modima_loggedout_page.html" >
        <head ></head>
        <body>
                <div id="main">
                        Welcome to Modima, The Mobile Direct Marketing Portal
                </div>
        </body>
</html>
```

Figure 4: The same page using DXe and an XML Component

The XML Component Solution

Docland's DXe software addresses the problem of inter-depended "include-files" by providing a more flexible and structured process for merging content from different sources. This process enables related content to be packaged as a single component, and provides a way to build pages by inheriting content from components in a manner similar to object-oriented programming.

Additionally, DXe/AJAX is a client side assembly process so it enables static components to be cached locally in the browser. This reduces network traffic and shifts page assembly workload from expensive server hardware into the users computer

Benefits of XML Components

- Accelerate web site development
- Improve consistency of web application presentation
- Improve quality & reliability by reducing project size
- Reduce network traffic by means of client-side component caching
- Reduce server load & transfer the cost of page assembly work from expensive server resources to your user's browsers
- Gain productivity by using "Object Oriented" processes for web page design
- Reduce maintenance costs & enable rapid, consistent site updates by eliminating content duplication

DXe implements a rules-based process for merging XML content. XML documents adhere to stricter rules of structure than HTML. This makes it simpler and faster to process XML content. DXe uses this innate structure of XML content to merge web pages.

XML Components

XML Components are the building blocks that DXe uses to build web pages. Each XML Component may itself be a complete XML document, such as a complete XHTML page. Each component may be developed and tested independently. DXe can also extract individual elements from XML files and utilise these elements as components in the page assembly process.

How DXe Works

XML Components are assembled into pages by merging the Document Object Model (DOM) trees of the XML files. This is achieved by identifying common elements within the files and merging the trees with reference to these common elements.

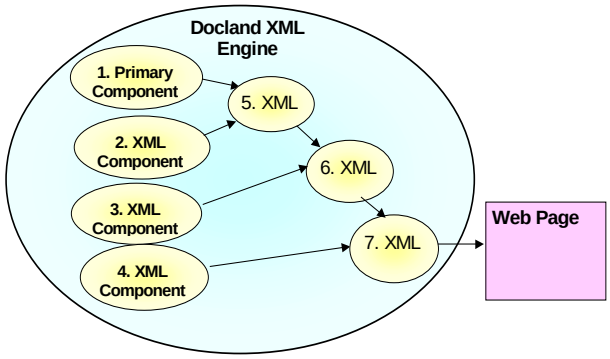

Figure5: DXe generates web pages by merging XML Files.

Figure 5 above illustrates how DXe merges XML Components to generate a web page. Each component, (1,2,3,4) is merged in turn to create intermediate documents (5 & 6 & 7). The last XML document (7) is converted to an HTML document (8) and displayed by the browser.

DXe assembles XML Components by merging their **hierarchical structures**. Merging is performed in two stages: Firstly DXe generates composite tags by **combining the matching tags** in each of the input component. This allows each component to contribute some or all of the attribute values that are required to product the complete XML or HTML tag in the finished page. Secondly, DXe **merges the descendent elements (children) of each matching element**, to produce a single element containing all of the children of both input elements. This feature allows each component to add some content to the final page.

The merging process preserves the structure of one of the input components and merges the ENTIRE content of the other into this structure. This procedure for merging components provides a meaningful method to merge the contents of XML documents.

DXe merges components according a set of simple rules. DXe also provides a small set of mark-up tags and attributes that allow the developer control the merging process. A patent is pending on this merging process.

DXe in Web Applications

DXe enables web application developers to develop the presentation layers of web applications using a component-based approach. DXe complements the presentation layer facilities of server-side frameworks such as Struts.

Figure 6 below shows a typical 3 Layer Java Server application built using DXe.

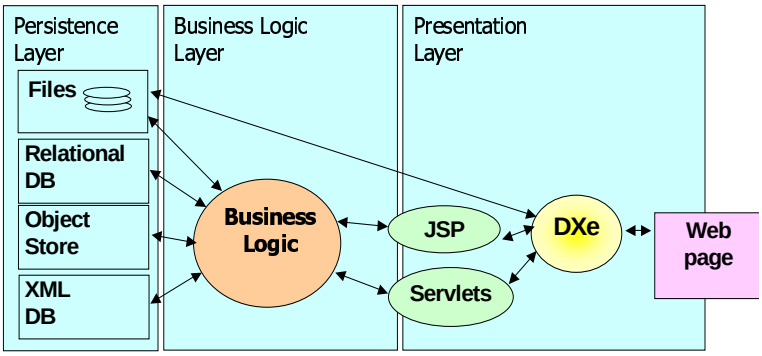

Figure 6: DXe in the presentation layer of a web application

The Docland XML Engine (DXe), when compared to other template-based methods of separating content from presentation has the following advantages:

1. **Unobtrusive**
   DXe can be added to existing applications with minimal modification of the systems JSPs and servlets. Client-side DXe can be used with static web content..

2. **Pure (x)html**
   DXe components are pure XHTML that can be viewed using html editors. No scripting is required (DXe/AJAX library must be added to web pages to enable client side assembly if required).

3. **Simple to use**
   The concepts behind DXe are few, and relatively straightforward.

4. **Follows O-O inheritance principles**
   DXe follows object-oriented inheritance principles, which makes their design a logical extension of code design. Large-scale changes can be made to entire enterprise web identities by editing a very small number of files.

An Example

This example shows how DXe can be used to generate the login and registration pages of a web site.

Description

This web application has a login form and a registration form. These forms are implemented as Java Server Pages (JSPs). To log in to the application the user enters a username and a password. To register a new user must enter a unique username, enter a password into two input boxes, and select one of two possible user types.

Figures 7 & 8 show the login and registration forms for this example application.

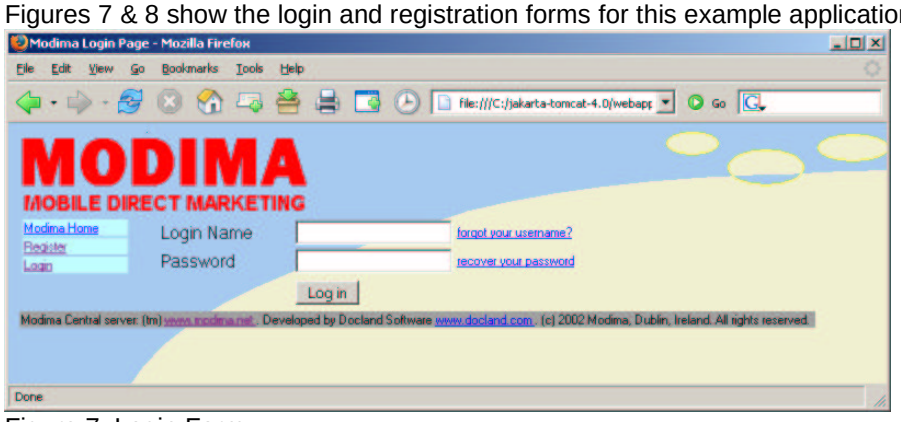

Figure 7: Login Form

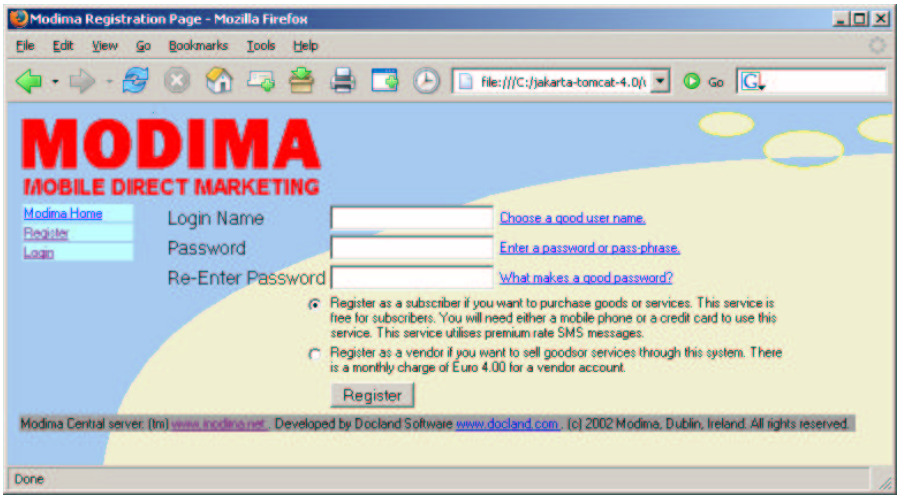

Figure 8: Registration Form

It is obvious from the figures above that these forms share a great deal.

They share the general format of the page - the background graphics, a table structure for the layout, and the menu with 3 items. They both also contain a "Login Name" input box, and a "Password" input box.

These forms also differ in a few significant ways:

- The window titles are different
- The help messages displayed beside the Login name and pasword boxes are different
- The register form has an additional password text box.
- The register form contains additional radio buttons and help text
- The "submit" button has a different text value in each form.

Using DXe in this Example

DXe can be used very effectively in this example. The objective in this scenario is to create a presentation layer that is flexible and maintainable.

- **Flexibility** means that the presentation layer can be easily manipulated to take different forms and layouts depending on runtime conditions (e.g. the browsing device geometry, the user's choice of language, or personal attributes such as visual impairment).

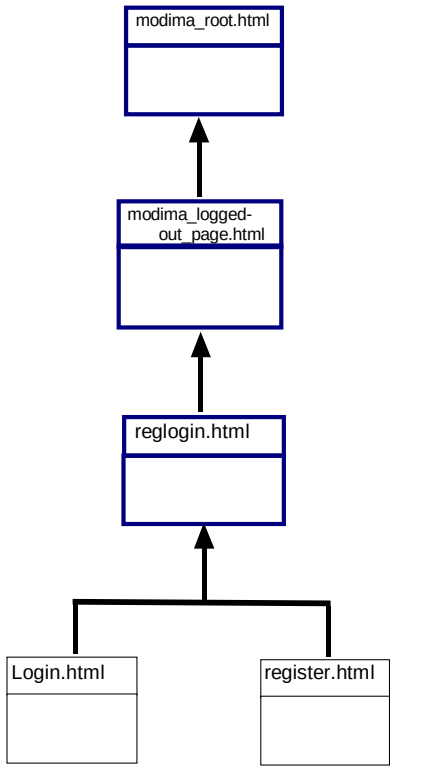

Figure 9: A simple component model for Login and Register pages

- **Maintainability** requires that duplication of content is eliminated, that any relationships between various elements of the presentation are obvious and explicit.

These objectives are achieved by building the web pages from XML components. A component-based architecture eliminates duplication, makes the system easier to maintain, and enables developers to deliver greater variety in the presentation layer.

DXe enables a developer to build a component model for the applications web pages. The structure of this model will be determined by the requirements of the system being developed.

This example will illustrate how DXe can be used to factor the login and register pages into two sets of components: one set of components that encapsulate the common elements of these two pages; and another set that encapsulates the characteristics unique to each page.

The application will rely on DXe to build each web page by combining the appropriate components at run time.

In a conventional implementation, JSPs would generate the web pages for this system. JSPs provide only very rudimentary support for content reuse through the use of include-files.

Figure 9 shows how these pages can be factored into separate components. In this example, login.jsp generates the dynamic content for the login page and also contains static markup unique to the login page and inherits other shared content from regLogin.html. RegLogin.htm contains most of the layout information for the register and login pages, and it inherits from openManu.html and modima.html.

The register page is constructed in a similar way to the login page. This simple approach factors the pages into the following components:

Set 1: Reusable components

modima_root.html: this component contains the basic layout of the page, including the menu on the left hand side of the page, one item on the menu ("Modima Home"), and the copyright notice at the bottom of the page. This component is the root component for the site, so every page in the application follows this structure and contains the menu, "Modima Home" menu item and the copyright notice.

modima_loggedout_page.html: this component contains the menu items that are available when a user is not logged in. These are the options "Register" and "Login". These options are placed in a separate component so that thy can be added or omitted from the menu as a unit. Once a user has logged into the system, this component is omitted from all web pages, but replaced by a more relevant menu of options - which would also be an XML component, or set of components.

reglogin.html: this component contains everything that is common to the two pages: such as the form, the tabular layout within the form, the "Login Name" and "Password" prompts and input boxes. The XML source of this component is shown below. Note the dxe:extends attribute of the HTML tag. This indicates the components that RegLogin.html inherits from.

Note also that the form tag has no "action" attribute and that the submit button does not have a "value" attribute. These attributes are not common to the two pages that inherit from regLogin.html (the register page and the login page), so they are not included in this component. The appropriate values for these attributes are included in the components register.html and login.html.

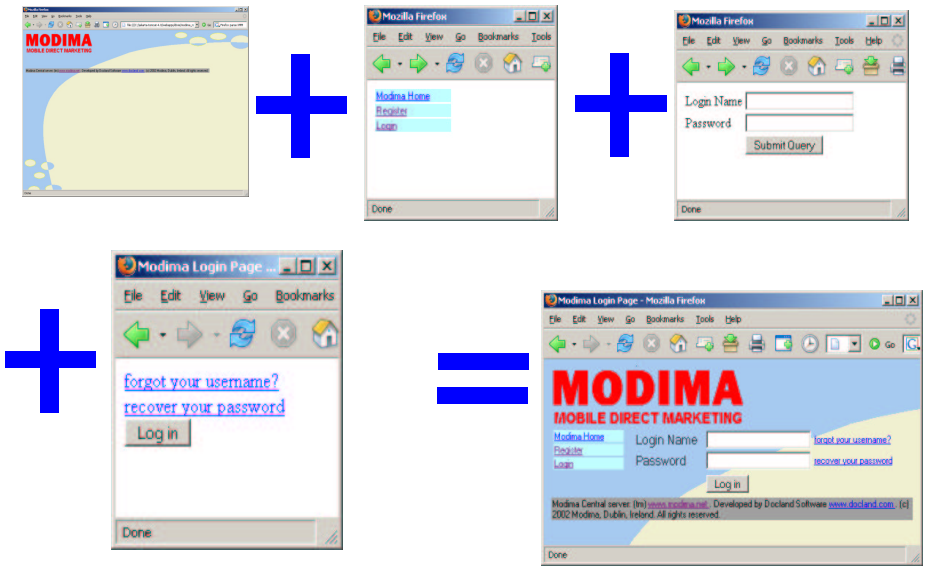

Figure 10: Illustration of components assembled into the login page

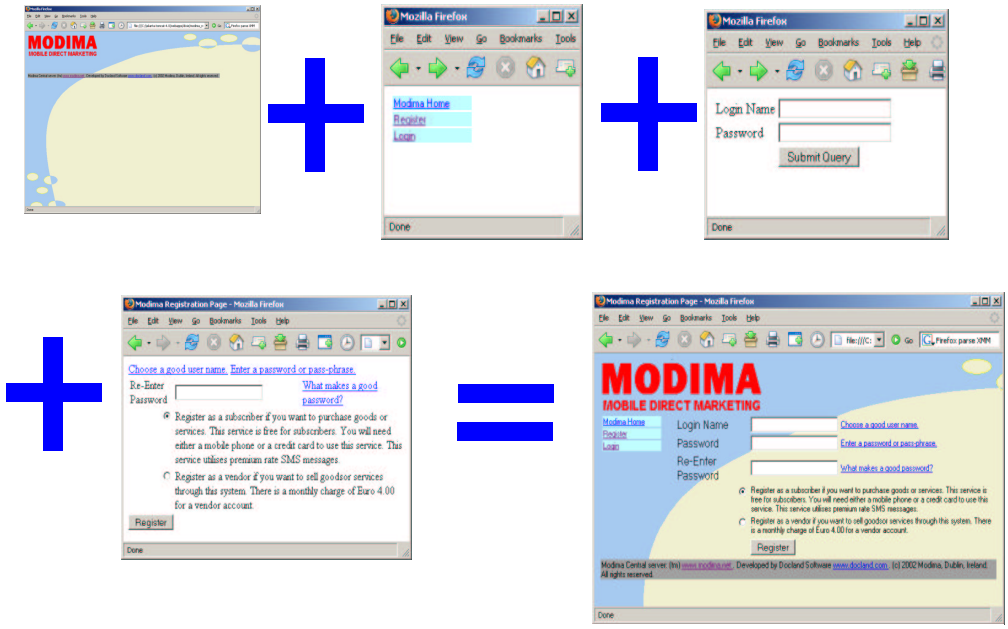

Figure11: Illustration of components assembled into the registration page

Set 2: Unique Content

Login.html: this component contains everything that is unique to the Login form - the form title, the help text & links for the username, password and the "Login" label for the submit button.

register.jsp: this component contains everything that is unique to the register form - such as the form title, the additional password box, the radio buttons and the "Register" label for the submit button.

See this example in action at http://www.modima.net.

Client-side Assembly & Content Caching

DXe provides a mechanism to assemble web pages from components of XML. Any XML document can be factored into a set of XML components. DXe can reassemble these components to generate the original XML document. DXe therefore offers great potential for use in content caching systems. Large web pages often contain considerable amounts of static information and relatively little dynamic content. Caching static content as XML components can reduce network traffic and processor overhead. DXe enables dynamic pages to be generated more efficiently because back-end business logic need only deliver the important dynamic information in response to any request. DXe can then generate the full page using cached copies of static components.

DXe can also remove from business logic layers the task of assembling dynamic content from different sources into web pages.

Additionally, DXe assists application developers to provide alternate presentations for web applications. This is important for wireless applications, where screen size and network bandwidth may be restricted. Using DXe, application developers can quickly create alternative presentation layers tailored to a particular type of client device.

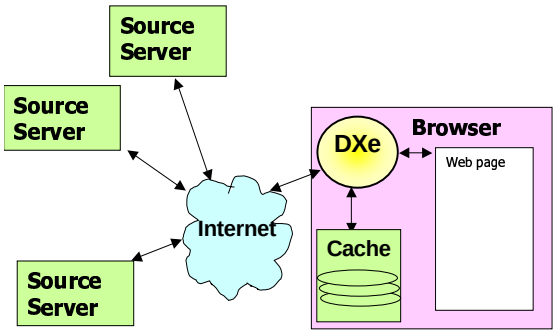

Figure 12: Client-Side Assembly using DXe

More Information about DXe

For more information about Docland Software or DXe, please contact :
Garrett O'Carroll, garrett@docland.com
Phone: +353-86-2642801.
Website: www.docland.com

Patents granted: USA: 6772165, also in other countries.

Document id: DXEWP60 DXe White Paper revision 6.0 7/6/06