韓國컴퓨터情報學會 論文誌
第15卷 第7號, 2010. 7.

2010-15-7-4-5

CSS 가독성 향상을 위한 최적화기법

정 우 성*, 이 은 주**

An Optimization Technique to Improve Readability of CSS

Woosung Jung*, Eunjoo Lee**

요 약

웹 어플리케이션의 원활한 유지보수를 위하여 소스 페이지의 가독성 향상은 필수적이다. CSS는 웹 페이지의 구성 요소 중에서 순수한 표현계층이지만 인라인 형태로 삽입되는 경우가 많고, 웹 개발도구에 의해 자동생성되기도 하는데, 이들은 전체 코드의 가독성과 UI 계층의 재사용성이 떨어지게 된다. 기존의 CSS 최적화 연구는 주로 사이즈 압축을 다루고 있어 재사용성이나 가독성에 초점을 맞추고 있지 않다. 본 논문에서는 CSS 가독성 및 재사용성 향상을 위하여 CSS 코드를 구조화하고, 가독성 향상을 위한 기준을 정의하였다. 이들을 기반으로 최종적으로 CSS 코드의 가독성을 높이는 알고리즘을 제안하고, 예제 및 실험을 통하여 본 접근법의 유용성을 보인다.

Abstract

For effective maintenance for web applications, it is necessary to improve the readability of the source pages. Though CSS(Cascading Style Sheet) belongs to pure presentation layer in various web constituent entities, CSS codes are often used by inlning and they are sometimes automatically generated by web development tools. The existing studies on CSS optimization have only focused on reducing the size of codes and they did not incorporate the readability or the reusability. In this paper, CSS codes are structured and several criteria for readability are defined to improve the readabilty and reusability. Based on them, the algorithm to improve the readability are proposed. Finally, case study are presented to show the applicability of the proposed algorithm.

▸ Keyword : CSS(Cascading Style Sheet), 가독성(readability), 웹 어플리케이션(web application), 최적화(optimization), 웹 리팩토링(web refactoring)

• 제1저자 : 정우성 교신저자 : 이은주
• 투고일 : 2010. 04. 30, 심사일 : 2010. 05. 18, 게재확정일 : 2010. 05. 30.
* 서울대학교 공과대학 컴퓨터공학부 박사과정 ** 경북대학교 IT대학 컴퓨터학부 조교수
※ 이 논문은 2007년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업 연구임 (NRF-2007-331-D00407)

Ⅰ. 서 론

현재 웹에 기반한 소프트웨어 개발이 점차 증가하고 있으나 사용자들의 만족도는 높지 않다 [1]. 왜냐하면, 웹 어플리케이션은 타 소프트웨어에 비하여 변경 요구가 잦으며 그 생명주기가 짧기 때문이다. 이러한 웹의 특성으로 인하여, 웹 개발은 높은 유지보수성을 요하며, 가독성 향상은 그 방안 중의 하나이다. 스타일 시트(Style Sheet)는 구조와 콘텐츠를 표현으로부터 분리하는 방안이며[2], CSS(Cascading Style Sheet)[3]는 대표적인 스타일 시트로서 현재 웹에서의 핵심 기술이다[4]. 그런데 클라이언트측의 UI관련 코드들 중에서 CSS 는 다른 HTML, 스크립트 및 서버측 코드들에 비해 순수한 표현 계층임에도 불구하고 인라인(inline) 형태로 삽입되는 경우가 많다. 이러한 인라인 기법은 웹 페이지를 기타 요소들에 종속적이고, 물리적 독립성을 보장하기 어려운 형태의 구조로 만들게 되며, 결국에는 전체 코드의 가독성과 UI계층의 재사용성을 떨어뜨리는 원인이 된다. 뿐만 아니라, CSS 코드의 경우는 웹 개발 도구에 의해 스타일을 적용하는 과정에서 자동 생성되는 경우가 대부분이지만 최적화가 이루어지지 않아서 CSS코드가 중복되거나, 불필요한 내용이 포함되는 경우가 많다.

기존의 CSS 최적화 관련한 접근법들이 존재한다 [5][6][7][8]. 이들은 대부분 사이즈를 줄이기 위한 압축에만 초점을 두고 있어, 가독성 및 재사용성 향상에 있어 효과적이지 못한 편이며, 현재 CSS의 가독성 향상에 초점을 맞춘 연구는 부족한 실정이다. 따라서 본 연구에서는 CSS의 가독성 향상을 위한 프로세스를 제안한다. 이를 위하여 CSS의 확장된 표기법을 제안하고 최적화 과정을, 비교적 기계적으로 수행 가능한 단순 최적화와, 최적화 알고리즘의 적용이 필요한 내부 최적화로 구분하였다. 단순 최적화는 내부 최적화를 위한 일종의 전처리 작업으로 중복을 줄여서 가급적 필요한 코드만을 남겨두려는 것으로, 기존의 사이즈 압축과 관련이 있다. 이후 정리된 코드를 대상으로 가독성 향상을 위하여 UI와 구조와의 결합도를 낮추는 알고리즘이 적용된다. 본 논문에서 제안한 기법은 일종의 웹 리팩토링(web refactoring) 기법으로서, 표현계층인 UI의 변화 없이 CSS의 구조 및 가독성을 향상시키는 접근법이라고 할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구의 배경에 대해 살펴본다. 3장에서는 본 논문에서 제안한 CSS 최적화 프로세스에 대해 정의하고 CSS의 구조 평가 기준에 대해 기술한다. 4장에서는 최적화 프로세스에 대한 사례 연구를 보이며 이를 위해 새롭게 정의된 표기법에 대해 언급한다. 최종적으로 5장에서 결론을 맺고 향후 연구를 제시한다.

Ⅱ. 연구 배경

CSS는 웹 문서에 폰트나 색깔과 같은 스타일을 부가하는 기술로서 [3], 웹 페이지의 UI 계층에서 중요한 비중을 차지한다. CSS의 최적화와 관련하여 기존에 존재하는 접근법 [5][6][7][8] 은 학문적인 관점이라기보다 사이즈 압축 등을 위한 실용적인 관점에서의 유용한 팁 수준이다. [5]에서는 CSS만 단독으로 검사하여, 압축하여 이전과 이후의 압축량을 비교해준다. 그렇기 때문에 HTML에 실제로 사용되지 않는 CSS 정의라고 하더라도 그냥 남겨둔다. 태그(tag)와의 의존성(dependency)를 고려하지 않으며 주로 동일한 속성(property)과 값(value)를 가진 것들을 묶는 방식으로 압축한다. 이는 속성의 개수가 적을 때는 적합하지만, 그 수가 늘어날 경우는 좋은 결과를 얻기 어렵다. 그리고 CSS의 값 등이 다르더라도, 이를 처리하지 않고 단순한 스트링으로 계산해 버린다. 뿐만 아니라 문법(syntax)이 달라도, “속성:값”의 쌍만 비교하여 찾기 때문에 정확한 결과를 구하기 어렵다. [6]에서는 몇 가지 규칙을 적용해서 CSS의 사이즈를 줄이며, 이들 규칙은 옵션으로 처리가능하다. 규칙의 예를 들면, 0인 경우 단위를 생략한다거나 동일한 규칙이나 선택자(selector)는 연결하고, 중복 속성은 제거하며 margin이나 padding등 의미 없는 값은 함께 제거를 한다. 또한 결과에 대해서도 통계를 보여줄지, 색으로 토큰을 보여줄 지 등을 선택할 수 있다. 하지만 이것 역시 CSS의 규칙만으로 압축을 하고 있다. [7]의 경우 [6]와 유사한 방식으로 동작하며 결과도 유사하다. Clean CSS [8]는 syntax에 대해서도 체크를 하며, 이전 연구들과 유사한 방식으로 압축을 한다. 선택자와 속성을 3단계로 압축할 수 있도록 하며, 불필요한 ‘\’나 마지막 ‘;’, 주석(comment) 등을 기본적으로 제거한다. 비교적 다양한 옵션과 코드 레이아웃(code layout)에 대해서도 압축정도에 따라 다양한 방식을 지원하며, 탬플릿(template)을 이용하여 사용자 정의도 가능하다. 이들 접근법은 CSS의 가독성 향상보다는 사이즈 압축에 초점을 맞춘 것으로 오히려 압축 후에 가독성이나 재사용성은 떨어지게 된다.

기존의 CSS 최적화 연구는 대부분 실용적인 관점에서의 유용한 팁 수준으로, 학술적인 관점에서 접근한 연구들이 부족하다. [4]에서는 CSS 코드의 품질 우수성을 나타낼 수 있는 메트릭을 제안하였다. 이것은 추상성 요소(Abstractness-factor)로서 사용된 클래스 및 id 선택자에 대한 태그 선택자의 비율

이나 후손 선택자(descendant selector)의 양 등을 이용하며, 추상성이 높을수록 콘텐츠와 표현의 분리 정도가 높아 재사용성이 높다고 판단하였다. 이를 이용하여 자동 생성된 CSS코드보다 수동으로 작성한 코드의 품질이 우수함을 보였다. 그 외 CSS 문서 내의 선언 개수를 이용한 코드의 효율성(efficiency)이나 생성된 CSS코드의 출력 양식(formatting)을 이용한 가독성에 대하여 간단히 언급하였다. [4]에서는 본 연구와 마찬가지로, CSS 코드 생성을 코드의 효율성 및 추상성 요소를 평가함수로 활용한 최적화 문제로 보고 있으나, 메트릭 제안이 주요 목적이며 CSS 코드 생성은 범위에서 제외되었다.

III. 클러스터링 기반 CSS 최적화 프로세스

3.1 CSS 구조 평가 기준 - 결합도

CSS는 웹 어플리케이션의 UI계층을 형성하고 있으며, 그 구조에 대해 가독성, 유지보수성, 재사용성, 최소성 등의 품질 속성들을 평가할 수 있다. 일반적으로 CSS의 구조는 결합도(coupling), 중복된 코드 비율, 불필요한 코드 비율, 전체 CSS 코드의 크기 등으로 평가할 수 있다. 일반적으로 결합도는 비교대상 모듈들 간의 연결 강도를 말하며, 낮을수록 해당 소프트웨어의 품질이 높다고 알려져 있다[9]. 웹 기반에 대해 [10][11]에서 정의된 웹 페이지 사이의 결합도가 존재하지만, 본 논문에서의 결합도는 CSS와 페이지의 다른 구성요소들 사이의 긴밀함을 보여주는 척도이다. 특히 디자인인 CSS와 구조인 HTML 사이의 연결강도를 의미한다[12].

자동 생성된 CSS 코드의 경우 대부분 결합도는 낮지만 중복된 코드나 불필요한 코드의 비율이 높고, 이로 인해 CSS 코드의 분량이 높아질 뿐 아니라 복잡도가 높아서 이해하기 어렵게 된다[4]. 또한 유지보수를 하는 과정에서 인라인 형태의 CSS 코드가 삽입되는 경우가 많고, 이로 인해 결합도가 높아지게 되며, 독립적으로 분리시키기 어려운 UI계층 구조가 만들어지는 경향이 있다. 실제로 결합도는 CSS가 삽입되어 있는 형태와 정의 형태의 2가지 요소에 영향을 많이 받게 된다. 먼저 삽입 형태에 대해서는 link 태그 또는 @import를 이용하여 외부 파일로 분리시켜 놓는 경우가 가장 낮고, style 태그를 이용하여 기타 요소들과 독립은 시키되 동일한 파일에 포함된 경우가 중간 정도이며, HTML 태그나 Script 등에 삽입(inline)되어 있는 경우가 가장 높다. 정의 형태의 경우는 class나 id를 선택자(selector)로 이용하는 경우가 낮고, html의 구조에 따라 정의한 경우는 높게 나타난다. 또한 도구를 통해 자동화시킨 경우는 최적화가 생략되고, 재사용성을 높이기 위해 불필요하거나 중복된 코드들이 삽입되어 있는 경우가 많은데, 그러한 경우가 아니라도 specificity나 inheritance, cascade 등에 의한 영향이나 LVFHA()의 경우처럼 우선순위에 의해 효과가 사라지는 경우도 있다[13].

본 연구에서는 단순 최적화 단계에서 HTML 태그나 Script 등에 삽입되어 있는 CSS의 정의를 클래스로 고유 id를 부여하여 분리시키는 과정을 거친다. 이 과정에서 HTML 문서와 CSS 정의와의 결합도를 강제적으로 떨어뜨리게 된다. style 정의의 길이가 긴 경우는 대부분 class 레퍼런스로 변경되기 때문에 HTML 코드의 길이가 줄어듬과 동시에 style 속성의 개수가 줄어들게 된다. 강한 결합도는 style 속성의 개수와 관련이 많기 때문에, 태그 개수 대비 style 속성의 개수의 비율을 결합도 판단 기준으로 삼을 수 있다. 그 이후의 내부 최적화는 분리된 CSS 정의 집합에 대한 재사용성 및 가독성과 관련한 최적화로 결합도와는 독립적이다.

3.2. CSS 최적화 프로세스

기존에 존재하는 연구들은 2장에서 설명하였듯이 CSS 구조 향상을 위하여 중복되거나 불필요한 코드 비율이나, 전체 CSS 코드 크기를 줄이는 데 초점을 맞추고 있다. 그러나 본 논문에서는 가독성 및 유지보수성 향상을 목적으로 하고 있으며, 이를 위하여 가장 효과적인 방법 중 하나는 웹 페이지의 디자인과 구조 정보를 분리시킴으로써 CSS와 HTML과의 결합도를 떨어뜨리는 것이다. 이를 위한 최적화 프로세스는 크게 단순 최적화 단계와 내부 최적화 단계로 나눌 수 있으며, 아래 <그림 1>은 각각의 하부 단계들에 대한 개요 및 순서를 보여준다. 단순최적화는 사이즈를 줄이고 결합도를 떨어뜨리는 데 초점을 맞추며, 내부 최적화는 분리된 CSS 코드를 대상으로 가독성 및 재사용성을 높이는 데 초점을 맞춘다. 이후 함수 기반 최적화는 내부 최적화 이후에 연속되는 패턴을 다시 함수 형태로 묶어주는 기계적 과정으로 선택적으로 실시할 수 있다.

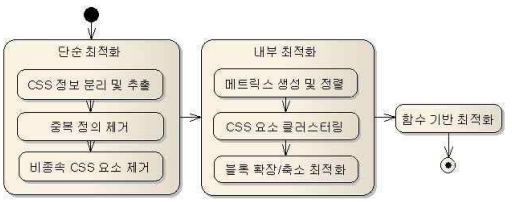

그림 1. 전체 프로세스 개요
Fig. 1. The Overall Process

3.2.1 단순 최적화

단순 최적화는 비교적 기계적인 과정으로, 내부 최적화를 수행하기 전에 적용한다. 본 절에서는 이러한 단계를 실제 상용 워드 프로세서를 통해 자동 생성한 웹 페이지 사례를 바탕으로 각 단계를 설명한다. 다음은 생성된 CSS 코드의 일부이다.

```
...
P.HStyle0, LI.HStyle0, DIV.HStyle0
        {style-name:"바탕글"; margin-left:0.0pt;
margin-right:0.0pt; margin-top:0.0pt;
margin-bottom:0.0pt; text-align:justify; text-indent:0.0pt;
line-height:160%; font-size:10.0pt; font-family:바탕;
letter-spacing:0.0pt; font-weight:"normal";
font-style:"normal"; color:#000000;}
P.HStyle1, LI.HStyle1, DIV.HStyle1
        {style-name:"본문"; margin-left:15.0pt;
margin-right:0.0pt; margin-top:0.0pt;
margin-bottom:0.0pt; text-align:justify; text-indent:0.0pt;
line-height:160%; font-size:10.0pt; font-family:바탕;
letter-spacing:0.0pt; font-weight:"normal";
font-style:"normal"; color:#000000;}
P.HStyle2, LI.HStyle2, DIV.HStyle2
        {style-name:"개요 1"; margin-left:10.0pt;
margin-right:0.0pt; margin-top:0.0pt;
margin-bottom:0.0pt; text-align:justify; text-indent:0.0pt;
line-height:160%; font-size:10.0pt; font-family:바탕;
letter-spacing:0.0pt; font-weight:"normal";
font-style:"normal"; color:#000000;}
P.HStyle3, LI.HStyle3, DIV.HStyle3
        {style-name:"개요 2"; margin-left:20.0pt;
margin-right:0.0pt; margin-top:0.0pt;
margin-bottom:0.0pt; text-align:justify; text-indent:0.0pt;
line-height:160%; font-size:10.0pt; font-family:바탕;
letter-spacing:0.0pt; font-weight:"normal";
font-style:"normal"; color:#000000;}
...
```

[단계1] CSS 정보 분리 추출

style 속성으로 HTML 중간에 삽입되어 있는 스타일 관련 정보들을 모두 CSS 정의 요소로 분리하여 추출한다. 이 과정에서 style 대신 class 속성이 삽입되고, 이를 통해 HTML로부터 CSS 정의를 분리시킬 수 있다. 하지만, 스타일 정의 및 class 속성이 추가되면서 전체 웹 페이지의 사이즈는 오히려 약간 증가하는 경향이 있다. 아래는 추가된 클래스 속성의 일부로 21개의 class가 삽입이 되었다.

```
<style>
.class1 {border-collapse:collapse;border:none;}
.class2 {border-left:solid #000000
0.4pt;border-right:solid #000000 0.4pt;border-top:solid
#000000 0.4pt;border-bottom:solid #000000
0.4pt;padding:1.4pt 1.4pt 1.4pt 1.4pt}
.class3 {border-left:solid #000000
0.4pt;border-right:solid #000000 0.4pt;border-top:solid
#000000 0.4pt;border-bottom:solid #000000
0.4pt;padding:1.4pt 1.4pt 1.4pt 1.4pt}
...
```

[단계2] 중복 정의 제거

동일한 정의가 있다면 동일한 이름으로 묶어준다. 이를 통해 명시적인 CSS 중복 정의를 제거한다. 아래는 중복이 제거된 코드 일부이다.

```
...
<style>
.class1 {border-collapse:collapse;border:none;}
.class2 {border-left:solid #000000
0.4pt;border-right:solid #000000 0.4pt;border-top:solid
#000000 0.4pt;border-bottom:solid #000000
0.4pt;padding:1.4pt 1.4pt 1.4pt 1.4pt}
.class6 {font-weight:"bold";}
.class12{color:#0000ff;}

</style>
...
```

[단계3] 비종속 CSS 요소 제거

해당 웹 페이지에 종속되지 않은 CSS 정의 요소들은 불필요하므로 모두 제거한다. 이러한 불필요한 정의들은 웹 페이지를 도구에 기반하여 자동 생성했을 때 나타나는 경우가 많다. 이 작업은 큰 비용이 들지 않기 때문에 기계적인 방법으로 쉽게 처리할 수 있다. 아래 코드는 <단계 2>에서 제거된 코드의 일부이다.

```
...
P.HStyle7, LI.HStyle7, DIV.HStyle7
        {style-name:"개요 6"; margin-left:60.0pt;
margin-right:0.0pt; margin-top:0.0pt;
margin-bottom:0.0pt; text-align:justify; text-indent:0.0pt;
line-height:160%; font-size:10.0pt; font-family:바탕;
letter-spacing:0.0pt; font-weight:"normal";
font-style:"normal"; color:#000000;}
P.HStyle8, LI.HStyle8, DIV.HStyle8
        {style-name:"개요 7"; margin-left:70.0pt;
margin-right:0.0pt; margin-top:0.0pt;
margin-bottom:0.0pt; text-align:justify; text-indent:0.0pt;
line-height:160%; font-size:10.0pt; font-family:바탕;
letter-spacing:0.0pt; font-weight:"normal";
font-style:"normal"; color:#000000;}
P.HStyle9, LI.HStyle9, DIV.HStyle9
        {style-name:"쪽 번호"; margin-left:0.0pt;
margin-right:0.0pt; margin-top:0.0pt;
margin-bottom:0.0pt; text-align:justify; text-indent:0.0pt;
line-height:160%; font-size:10.0pt; font-family:굴림;
letter-spacing:0.0pt; font-weight:"normal";
font-style:"normal"; color:#000000;}

...
```

아래 <표 1>은 각 단계별 단어 및 글자 수의 변동을 보여준다. 3단계까지 처리한 결과, 실제 페이지의 크기는 초기에 비해 워드 수 기준으로 38.3%, 문자 수 기준으로 31.3%로 줄어들었다. 대부분이 2~3단계에 줄어든 것이며, 이는 CSS 자동 생성 과정에서 중복되거나 불필요한 스타일 요소들이 많이 포함되어 있다는 것을 보여준다.

표 1. 단순 최적화 시 단어 및 문자 수
Table 1. Word # and Character # in Simple Optimization

단계	단어 수	문자 수
변경전	605	8869
1단계	649	9252
2단계	461	6744
3단계	232	2779

또한, 단순 최적화 전에는 16개의 style 정의가 HTML에 삽입되어 있었지만, 결합도를 떨어뜨린 이후에는 HTML에 집적적으로 포함된 style정의가 0개로 줄어들었기 때문에, 결합도가 단순 최적화 이후에 보다 느슨해졌다고 판단할 수 있다.

3.2.2 내부 최적화

이 단계는 CSS 정의 요소 내부의 중복요소를 최소화하기 위한 것이다. 단순 최적화의 [단계3]에서 상당히 많은 정의요소들이 의존관계에서 누락되어 제거되었지만, 대부분의 경우는 최종적으로 남은 CSS 정의 요소들 간의 유사성으로 인해 추가적인 최적화가 필요한 경우가 많다. 이는 클러스터링과 관련한 최적화 문제이다. 본 연구에서는 스타일의 정의를 선택자에 따른 "속성:값" 의 집합으로 본다. 여기서의 선택자는 특정 웹 트리를 지정하는 일종의 포인터에 해당하며, 그 대상에 대한 UI특성을 각 속성들에 대한 값으로서 나타낸다.

아래는 [단계3]에서 제거되기 전에 가지고 있던 모든 CSS정의요소들을 "속성(property):값(value)"의 집합으로 나타낸 것의 일부이다. 예를 들어 위의 CSS코드에서 "font-style:9.0pt" 에서처럼 font-style이 속성(property)이며 9.0pt가 값(value)에 해당한다. 차이가 나는 부분은 색이 표시되어 있다. 하지만, 중복된 정의가 존재하며, 이 경우는 중복을 묶을 수 있는 다양한 방법들이 존재하기 때문에 기계적인 방법으로는 접근하기 어렵다는 것을 알 수 있다.

```
"P.HStyle0, LI.HStyle0, DIV.HStyle0"

        style-name:"바탕글"                          margin-left:0.0pt
        margin-right:0.0pt                          margin-top:0.0pt
        margin-bottom:0.0pt                         text-align:justify
        text-indent:0.0pt                           line-height:160%
        font-size:10.0pt  font-family:바탕
letter-spacing:0.0pt      font-weight:"normal"
font-style:"normal"       color:#000000

  ... (중략) ...

"P.HStyle12, LI.HStyle12, DIV.HStyle12"

        style-name:"미주"                           margin-left:13.1pt
        margin-right:0.0pt                          margin-top:0.0pt
        margin-bottom:0.0pt                         text-align:justify
        text-indent:-13.1pt                         line-height:130%
        font-size:9.0pt   font-family:바탕
letter-spacing:0.5pt      font-weight:"normal"
font-style:"normal"       color:#000000

"P.HStyle13, LI.HStyle13, DIV.HStyle13"

        style-name:"메모"                           margin-left:0.0pt
        margin-right:0.0pt                          margin-top:0.0pt
        margin-bottom:0.0pt                         text-align:justify
        text-indent:0.0pt                           line-height:160%
        font-size:9.0pt   font-family:굴림
letter-spacing:0.5pt      font-weight:"normal"
font-style:"normal"       color:#000000
```

결국 CSS의 가독성 및 유지보수성 향상을 위한 문제는 기계적인 프로세스를 제외하면 CSS정의요소 상에서의 최적화 문제이다. 단순화를 위해 속성 및 값을 구분하지 않게 된다면 일종의 정렬된 스트링(string)으로 볼 수 있으며 스트링의 최소 단위는 1개의 "속성:값"으로 이루어져 있고 이는 전체 CSS 코드에서 유일하다. 본 논문에서는 이를 아래 <정의 1>과 같이 CSS 스트링으로 칭한다.

[정의 1] CSS 스트링

CSS 정의 요소는 유일한 식별자인 "속성:쌍"들로 이루어진 문자열로 볼 수 있으며 이를 CSS 스트링으로 정의한다.

[정의 2] CSS 전체 코드

CSS 전체 코드는 CSS 스트링들로 이루어진 집합이다.

예를 들어 CSS 선택자 a, b, c의 정의가 아래와 같다고 하자.

```
a { p1:v1;p2:v2;p5:v5; }
b { p1:v1;p3;v3;p5:v5; }
c { p1:v1;p4:v4; }
```

위의 예에서 "p1:v1"과 "p5:v5"가 a, b에 공통으로 나온다면 이를 새로운 정의인 x로 치환하게 되면 a와 b에는 {p1:v1, p5:v5}대신 x가 들어가게 되며, 전체 CSS 정의 요소 중에 x의 비율이 높다면, {p1:v1, p5:v5}가 산재해 있는 것 보다 CSS 코드에 대한 재사용성과 가독성이 향상될 것이다. 즉, 대치(replace)한 코드(예에서의 x)의 출몰빈도가 높을수록 재사용성이 높고, 그리고 대치된 코드(예에서의 p1:v1, p5:v5)가 길수록 가독성이 높아진다고 볼 수 있다. 결국, 최대한 한꺼번에 많은 CSS의 공통 요소들을 묶을 수 있도록 내부 최적화를 하는 것이다. 하지만, 그러한 공통 요소를 묶는 방법은 한 가지만 존재하는 것은 아니며, 여러 가지 요소들에 대해 한 가지 기준만으로 커버할 수 있는 것도 아니다. 그러므로, 한 개 이상의 부분 스트링을 이용하여 최대한 많이 커버하는 것 역시 재사용성을 높이는 방법이 된다. 하지만, 이러한 부분 스트링이 크기가 너무 작고 개수가 많을 경우에는 오히려 가독성을 저해할 수 있기 때문에 커버 영역을 넓히되, 가급적 그 개수를 최소화 해야 한다. 이러한 서브 스트링은 실제 구현에서는 일종의 메트릭스로 표현되며, 이에 대해서는 4장의 사례연구에서 보다 상세히 다룬다. 따라서, 내부 최적화를 위한 기준은 아래와 같이 정리할 수 있다.

n개의 정렬된 CSS스트링에 대해
1) 최대한 많은 스트링에 포함되는 부분 스트링을 찾고
2) 그 부분 스트링은 가급적 길이가 길면 좋다.
3) 또한 최대한 많은 스트링을 커버하기 위해 복수 개의 부분 스트링을 이용할 수 있다.
4) 하지만, (3)에서의 부분 스트링의 개수는 최소화 할수록 좋다.

위의 예에서 a, b, c를 하나의 클러스터로 보고 “속성:값”을 각각의 유일한 식별자(unique identifier)로 보면, 선택자는 일종의 벡터로 표현이 가능하다. 벡터의 요소의 순서를 p1:v1부터 p6:v6까지로 보게되면 a의 경우 <1, 1, 0, 0, 1>의 값을 가진다. 즉, 모든 종류의 속성 및 값의 쌍인 최소 단위에 대해 유일한 ID가 부여되고, 속성에 대해 정렬될 수 있다고 가정하게 되면, 전체 n종류의 속성 및 값의 쌍이 있을 때, CSS의 선택자는 아래 <정의 3>와 같은 벡터로 표현할 수 있다. 여기서 pvk는 해당 쌍을 가지는지의 여부에 따라 0 또는 1의 값을 가진다.

[정의 3] CSS 정의요소 벡터 표현

d = <pv1, pv2, ..., pvn>

CSS의 정의가 CSS 파일에서 M개 존재하고, “속성:쌍”이 N개 존재한다면, MxN의 메트릭스가 아래와 같이 정의된다.

[정의 4] CSS 코드의 메트릭스 표현

$$CSS = \begin{vmatrix} d_{11} \dots d_{1N} \\ \dots \ \dots \ .. \\ d_{M1} \dots d_{MN} \end{vmatrix}$$

여기서

$$d_{ij} = \begin{cases} 1, d_i\text{의 } j\text{번째 요소가 존재} \\ 0, else \end{cases}$$

(di: i번째 CSS 스트링)

내부 최적화는 우선, 동일요소 합계 테이블을 생성하고, 최대한 넓은 영역을 최소한의 블록으로 커버할 수 있도록 생성한 후, 마지막으로 요소의 차이를 무시한 상태로 블록을 축소하거나 확장함으로써 커버리지를 높히는 전략을 쓴다.

[단계1] 메트릭스 생성 및 정렬

단순 최적화된 CSS 정의를 대상으로 속성,값의 쌍으로 이루어진 벡터를 하나의 요소값으로 가지는 벡터를 생각할 수 있다, 이 벡터들을 클래스의 id값에 따라 묶어서 나열하면 메트릭스를 구할 수 있다. 이 메트릭스에 대해서 속성,값의 쌍을 기준으로 동일한 값을 가지는 항목들을 요소별로 합산한 후에, 내림차순으로 정렬하면 클래스 id들간에 중복이 많이 생기는 요소,값들이 우선적으로 정렬되는 메트릭스를 구할 수 있다.

[단계2] CSS 요소 클러스터링

이전 단계에서 얻어진 정렬된 메트릭스를 대상으로 동일한 요소들을 가지는 CSS 정의들끼리 해당 요소들만으로 묶게 된다. 그러면 메트릭스의 부분 요소들이 포함되는 클러스터링을 구할 수 있는데, 이는 재사용성이나 가독성을 높이기 위해 다른 심볼로 대체할 수 있다. 만들어지는 클러스터링의 넓이가 넓을수록 최적화의 효과가 커진다. 하지만, 무조건 크게 묶는 것보다는 전체적으로 많은 요소들을 커버하는 것 역시 중요하다.

[단계3] 블록 확장/축소 최적화

이전 단계는 엄격한 조건만으로 클러스터링을 실시한 결과를 보여준다. 하지만, 일부 요소나 정의 항목들이 틀리더라도 우선은 클러스터링에 포함시키고, 차이점만을 추가로 표시하는 것이 재사용성이나 가독성에 유리할 수 있다. 그러므로, 이전 단계에서 얻어진 클러스터링을 중심으로 열이나 횡을 1~2칸 정도 추가적으로 포함시켜서 더 좋은 결과를 얻을 수 있는지 확인해보는 것이다. 구체적인 예는 실험 평가에서 다루므로, 여기서는 생략한다.

각 단계에 대한 구체적인 수행방법은 다음 4장에서 설명한다.

IV. 실험 평가

본 장에서는 3장에서 소개한 CSS의 예제에 대하여 전처리를 끝낸 이후, 내부 최적화를 수행하는 과정과 실험 결과를 보인다. 또한, 최적화된 결과를 효과적으로 유지보수하기 위해 확장된 CSS 표기법을 보인다. 하지만, 여기서 보이는 표기법의 문법은 결정적이지 않으며, 스타일의 상속이나 각 요소들의 오버라이딩 개념을 기반으로 다양한 형태로 확장이 가능하다.

실험을 위해 데이터는 앞서 단순 최적화를 거쳐서 만들어진 결과를 기반으로 “속성:값”에 기준하여 정렬하여 만든 14*14 메트릭스를 이용하였다. 이는 최적화되기 전의 CSS는 해당 메트릭에 속한 스타일 정보를 표현하기 위해 196개의 속성 및 값의 쌍을 이용하고 있음을 의미한다. 값의 차이가 있더라도 속성이 동일한 열에 위치하여 배열하였으며, 내부 최적화를 위해 우선 동일한 속성과 값을 가지는 영역을 크기에 기반하여 계속적으로 묶는 방법으로 최적화를 하였다. 이러한 클러스터링 최적화가 끝난 후에 열이나 행 방향으로 메트릭스를 확장하더라도 속성의 값이 차이가 나는 정도가 충분히 낮은 경우는 해당 영역을 속성 단위로 다시 한번 묶어줌으로써 CSS의 사이즈를 줄이고, 가독성을 높힐 수 있도록 하였다. 단순 최적화의 경우는 그 방법이 기계적이고, 앞 장에서 보였으므로, 여기서는 내부 최적화의 사례를 다룬다.

4.1 내부 최적화

[단계1] 메트릭스 생성 및 정렬

2개 이상의 동일한 속성 및 속성값을 가지는 요소들의 개수를 합하여 <표 2>와 같이 동일 요소 합계 테이블을 생성하고, 해당 테이블의 값이 큰 항목부터 정렬될 수 있도록 전체 메트릭스의 열의 순서를 변경한다. 사례에 사용된 스타일에는 총 14개의 스타일과 14개의 CSS 정의 클래스가 사용되었다. 이는 한글 2007에서 자동 생성한 HTML 문서를 기반으로 했으며, 확보된 14개의 스타일은 다음과 같다: style-name, margin-left, margin-right, margin-top, margin-bottom, text-align, text-indent, line-height, font-size, font-family, letter-spacing, font-weight, font-style, color. 그리고 이 문서를 자동생성하는 과정에서 정의된 CSS 클래스 정의 ID는 P.HStyle0 ~P.HStyle13까지 총 14개이다.

표 2. 동일 속성들의 연속된 개수로 정렬한 테이블
Table 2. The number of consecutive same properties in ascending order

margin-right	margin-top	margin-bottor	text-align	font-weight	font-style	color
14	14	14	14	14	14	14
N/A	N/A	N/A	N/A	N/A	N/A	N/A
text-indent	**font-family**	**letter-spacing**	**line-height**	**font-size**	**margin-left**	**style-name**
12	11	11	11	10	4	N/A
2	3	3	2	4	2	N/A

[단계2] CSS 요소 클러스터링

가로, 세로의 길이가 각각 2이상이면서 최대한 큰 영역을 차지할 수 있도록 사각형 영역을 찾되, 동일한 열에 속하는 속성 및 값은 모두 일치해야 한다. 앞선 테이블의 정보에 기반하여 높이가 14, 12, 11, 10 등의 경우에 대해 우선적으로 사각형을 찾을 수 있다. 일단 큰 블록이 고정되면, 해당 블록이 떨어지지 않는 조건하에서는 행과 열을 변경할 수 있다. 이는 이미 정해진 블록의 내부 항목끼리 순서를 바꾸거나, 외부 항목끼리 열이나 행의 순서를 바꾸는 것이 가능함을 의미한다. 하지만 내부와 외부의 행이나 열이 순서를 교환하게 되면, 사각형의 형태가 분리되기 때문에 이러한 정렬은 불가능하다. 이러한 제약 조건 하에서 열과 행의 순서를 변경하면서 블록으로 묶일 수 있는 또 다른 사각형들을 찾아서 계속적으로 묶게 된다. 이 과정에서 블록의 개수, 블록의 크기, 열과 행의 배열 상태 등에 따라 상당히 많은 경우의 수가 존재할 수 있다. 블록의 가로 길이가 긴 것은 해당 CSS 요소에 대한 표현을 그만큼 많이 줄여줄 수 있다는 것을 의미하고, 세로 길이가 긴 것은 많이 재사용될 수 있음을 의미한다. 그러므로 표현의 단순화 또는 재사용성 등 어느 것에 우위를 두느냐에 따라 클러스터링의 결과가 달라질 수도 있다. 하지만 이 둘을 모두 높이기는 어렵기 때문에 적당한 기준을 두는 것이 필요하다.

사각형의 가로 길이가 추상화되는 수준이므로 가독성, 사각형의 세로 길이가 추상화되는 대상의 범위이므로 재사용성에 해당한다고 볼 수 있다. 하지만 이 둘의 관계는 트레이드오프(trade-off)로, 어느 한쪽을 크게 하면 나머지가 줄어드는 경향이 있다. 그러므로 본 연구에서는 그 둘의 곱인 사각형의 영역을 기준으로 양쪽을 모두 충분히 고려해주기 위하여 사각형의 넓이를 우선으로 기준하였으며, 어느 쪽에 더 비중을 주느냐에 따라 그 기준식을 달라질 수 있다.

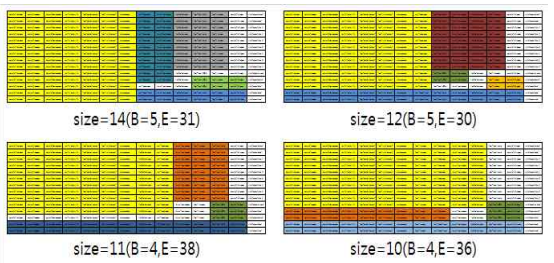

그림 2. 단계2 최적화의 예
Fig. 2. Examples of Optimization in Step 2

<그림 2>의 size는 초기 블록의 높이를 의미하고, B는 전체 블록의 개수, E는 블록으로 덮지 못한 영역의 개수를 의미한다. 블록의 개수가 적을수록 E가 큰 편이며, 동일한 블록인 경우라도 클러스터링의 형태에 따라 덮는 영역의 규모가 달라질 수 있음을 확인할 수 있다. 그러므로 2단계는 주어진 영역을 변화시켜서 최대한 적은 수의 블록으로 동일한 속성들을 묶어서 최대한 많은 범위를 차지하도록 하는 클러스터링 최적화 문제에 해당한다.

클러스터링을 위한 알고리즘에 따라 최적화의 결과가 달라질 수 있다. 본 연구에서는 블록을 구성하는 영역의 가로 길이를 X, 세로 길이를 Y라고 했을 경우 X>1, Y>1 인 조건을 만족하면서 남아 있는 영역을 최대한 많이 덮을 수 있는 사각형을 찾도록 행과 열의 경우의 수를 바꾸어 가면서 비교하였다. 초기 블록이 고정되면, 나머지 영역에 대해 유사한 방법으로 적용한다. 이 때 주의할 사항은 기존에 고정된 사각형의 영역이 분리되지 않는 범위내에서 행과 열의 변형이 이루어져야 한다는 것이다. B는 임계치를 줄 수도 있지만, 본 연구에서는 더 이상 X>1, Y>1인 블록 영역을 추가로 찾을 수 없을 때까지 진행하였다. 최종적인 평가는 가장 많은 영역을 덮으면서, 블록의 개수를 최소화할 수 있는 것으로 기준을 삼았다.

하지만, 3단계에서 추가적인 최적화가 이루어지는 과정에서 평가가 바뀔 수도 있기 때문에, 가장 큰 블록의 크기가 임계치(본 실험에서는 90) 이상 되는 결과에 대해서는 단계3을 적용하였다. CSS 정의의 규모나 내용에 따라 임계치의 기준이 다를 수 있기 때문에 명확한 기준이 있을 수는 없다. 하지만, 실험을 하는 과정에서 2단계 최적화의 결과의 순위는 매길 수 있으며, 상위로부터 어느 수준까지를 사용할지를 정의할 수는 있다. 임계치가 낮으면 계산할 가치가 없을 정도의 낮은 품질의 결과가 나올 수 있기 때문에 실험 비용이 많이 들고, 임계치가 높으면 다양한 해를 구할 기회를 놓치기 때문에 최선의 결과를 얻지 못할 수도 있다. 본 연구에서는 결과의 상위 10위 중에서 랜덤하게 4개(98(7*14), 96(8*12), 99(9*11), 110(11*10))를 추출하여 해의 다양성을 고려하여 임계치를 90으로 다소 낮추어 잡았다.

[단계3] 블록 확장/축소 최적화

전 단계에서 얻어진 블록들을 기준으로 주변부의 E영역을 덮을 경우 "속성:값"이 차이가 나는 개수가 충분히 적을 경우에는 오히려 블록을 재사용하고, 그 차이에 대한 부분을 오버라이드 하는 방법으로 CSS를 기술하는 것이 비용 효율적이다. 이렇게 함으로써 남겨진 E 영역을 최소화 할 수 있다. 허용하는 차이의 개수는 블록의 크기에 따라 틀려지지만 대부분의 경우 한 행에 1~2 정도가 적당하다. 아래는 범위 1 이내의 차이를 허용하도록 하여 최적화를 보완한 결과이다.

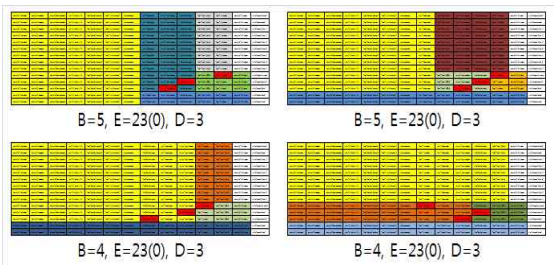

그림 3. 단계3의 최적화의 예
Fig. 3. Examples of Optimization in Step 3

2단계에서 주어진 블록을 기준으로 행이나 열을 인접한 영역까지 확장하거나 축소하여 최적화하였다. B는 블록의 개수, E는 블록이 커버하지 못한 영역, D는 블록이 확장되는 과정에서 발생한 전체 차이이다. 3단계에서는 새로운 블록이 추가되지는 않고, 단순히 블록을 확장 및 축소를 통해 E값과 D값을 최소화하는 방향으로 최적화를 시도한다. 만약, E,D값이 동일한 상태라면 E영역에 존재하는 "속성:값"의 쌍이 최대한 서로 상이한 것들만 남는 방향으로 최적화를 진행한다. 특히 E의 괄호값이 바로 유일한 값들의 개수와의 차이를 뜻하는데 0이라는 것은 E영역은 내에서는 동일한 속성,값 쌍이 존재하지 않음을 의미한다. E값이 0인 경우는 E 분포에 대한 최적화가 최대한으로 이루어진 상태에 해당한다. 그림과 같은 결과에 해당하는 최종 최적화 메트릭스의 집합들은 모두 E, D만으로는 같은 결과를 보이고 있다. 하지만, B가 낮은 경우가 더욱 좋은 평가를 얻게 된다. B까지 같은 값을 가지는 경우라면, 각 사각형들의 가로, 세로 길이에 대해 각각의 평균을 구해서 벡터를 만들 수 있다. 이 때, <그림 3>의 하단 좌측의 경우 (7.0, 6.5)이고, 하단 우측의 경우 (9.5, 4.0)이 된다. 벡터의 각 요소는 하나의 CSS 정의에 대해 얼마나 많은 요소들을 커버하느냐와 얼마나 많은 정의들을 묶어주느냐를 의미한다. 그러므로, 각각의 CSS 정의를 단순하게 보기 위한 가독성을 우선시 한다면, 벡터의 첫 번째 요소값이 높은 것을 선택하고, 전체 정의에 대해 일관성 있는 재사용성을 고려한다면 벡터의 두 번째 요소 값이 높은 것을 선택하면 된다.

본 연구에서는 클러스터링을 위한 재사용 단위를 미리 지정하였으나, 유전 알고리즘등을 적용하여 최적화를 할 수 있다. 최종적으로 최적화된 CSS는 표준 표기로 변환하거나 가독성이 더 높은 새로운 표기를 이용하여 유지보수 할 수 있다. 표기법에서의 구체적인 문법은 결정적이지 않으며, 클래스의 상속이나 오버라이드와 유사한 특성을 활용하여 확장하는 것이다. 최적화를 위한 확장된 표기법은 다음 4.2절에서 보인다.

4.2 확장된 CSS 표기법

본기존의 CSS 표기법으로는 2단계 상태의 최적화는 표현할 수 있지만, 3단계의 최적화는 지원하기 어렵다. 이를 위해 각각의 클래스 스타일 정의를 포함하고, 차이만을 추가적으로 표현할 수 있도록 CSS를 확장하였다. 여기서의 문법은 하나의 예일 뿐이며, 이러한 확장된 개념을 포함하도록 얼마든지 CSS를 확장할 수 있다. 다음은 14개의 클래스를 c1~c14로 명명하였을 경우 내부 최적화 3단계에서 얻어진 사례 4개 중 4번째에 해당하는 클래스 정의 중에서 일부를 보여준다. 초기 정의에 비해 전체 블록 스타일만 이해하고, 차이점만 파악하면 되므로, 훨씬 재사용성 및 가독성이 향상되었음을 알 수 있다. 지면상 블록의 정의는 생략하고, 가장 넓이가 큰 블록부터 B1~B4 라고 가정하였다.

```
c1 = B1+{margin-left:0.0pt;style-name:"바탕글"};
c2 = B1+{margin-left:15.0pt;style-name:"본문"};
...
c14 = B3+{style-name:"미주"};
```

실제 사용에서는 위의 표기에서 사용된 B1~B4라는 이름 대신 의미있는 이름을 사용하는 것이 바람직하다. 또한, c1~c10까지는 동일한 형태로 B1을 이용하고, margin-left와 style-name 만을 이용하고 있기 때문에, 함수화 과정을 통해 'c1 = B1+f(0.0,"바탕글")' 등으로 추가적으로 추상화 시킬 수 있다. 이러한 방법의 최적화는 기계적인 방법으로 구현 가능하다.

4.3 확장된 CSS 표기법

여전히 중복된 내용을 포함하고 있는 단순 최적화 상태와 내부 최적화를 한 후의 CSS 표기부분만을 비교해 보았다. 아래는 내부 최적화 후의 표기를 보여준다. 14개의 클래스를 각각 c1~c14로 명명하였으며, 속성과 값의 쌍의 비용을 1, 블록으로 표현되는 스타일의 비용을 s 로 하여 계산하였다. 다음 <표 3>은 비용의 전후 비율을 나타낸다. 앞서 얻어진 4번째 내부 최적화를 기준하여 계산하였다.

표 3. CSS 정의 비용 평가
Table 3. Evaluation of the CSS definition cost

class	cost rate	s=1	s=3	s=5
c1~c9	(s+2)/14	21.40%	35.70%	50%
c10~c12	(s+1)/7	28.60%	57.10%	85.70%
c13~c14	(s+1)/14	14.30%	28.60%	42.90%
average	(17s+26)/196	22.00%	39.30%	56.60%

블록은 스타일 속성 및 값 쌍들의 집합이긴 하지만, 그 결과는 어떠한 의미를 가질 수 있고, 반복해서 재사용될 수 있다. 그렇기 때문에, 초기에 해당 블록을 이해하기 위한 비용이 들 수는 있지만, 어느 정도 익숙해진 이후부터는 추상화된 개념으로 처리되기 때문에 속성 추가하는 것 이상의 큰 비용은 발생하지 않을 것으로 판단된다. 특히, 공간 비용의 경우는 s=1에 해당하며, 해당 사례의 경우 평균적으로 초기에 비해 22%로 줄어서 단순 최적화에 비해 CSS 표기 비용이 78% 감소된다. 이는 단순 압축이 아닌, 추상화를 통한 압축이기 때문에 각각의 스타일 정의를 이해함에 있어서도 훨씬 유리하다.

V. 결 론

본 연구에서는 CSS 코드의 가독성 및 재사용성 향상을 위하여 CSS와 HTML의 결합도를 우선적으로 낮추고, 분리된 CSS의 정의의 중복을 효율적으로 추상화시키는 방안을 제안하였다. 이를 위하여 전처리 작업에 속하는 단순 최적화와, 기계적 적용이 어려운 내부 최적화의 두 단계로 나누고, 필요한 평가 메트릭 및 CSS연산 표기법을 정의하였다. 그리고 실제 생성된 CSS 코드를 대상으로 적용하여 본 기법의 유용성을 보였다.

본 연구에서는 태그의 최적화를 고려하지 않고 있다. CSS 정의에서와 동일한 프로세스로 태그의 중복을 줄이고 재사용성과 가독성을 함께 높힐 수도 있다. 이 경우, 코드는 더욱 압축될 수 있고 추상화도 올라가게 된다. 이를 위해 CSS에서와 마찬가지의 표기법이 필요할 수 있으나 현재의 HTML 표준은 확장된 표기를 지원하지 않기 때문에, 유지보수 과정에서만 사용 가능하며 실제 배포되는 과정에서는 기존의 표기에 맞도록 기계적 코드 생성과정을 거쳐야 한다. 이렇게 함으로써, 개발자가 일반적으로 CSS를 다루는 동안은 가독성과 재사용성을 높이면서 기존의 표준과 호환을 이룰 수 있다.

기존의 웹을 재공학하기 위해서는 표준 CSS와 확장된 CSS간의 양방향 변환이 가능해야 한다. 확장된 CSS를 표준 CSS로 변환하는 것은 자명하기 때문에, 본 연구에서 보인 표준 CSS를 확장된 CSS로 묶어주는 방법을 통해 양방향 변환을 실현할 수 있다. 그와 별개로 CSS와 태그와의 의존성을 고려한 최적화 역시, 이론적 토대를 만들어 놓는 것이 필요할 것이다.

참고문헌

[1] 오성균, 김미진, "웹 어플리케이션의 복잡도 예측에 관한 연구," 한국컴퓨터정보학회 논문지, 제 9권, 제 3호, 27-34쪽, 2004년 9월.

[2] J. Korpela, "Lurching toward Babel: HTML, CSS and XML," IEEE Computer, Vol. 31, No. 7, pp. 103-104, 106, 1998.

[3] *http://www.w3.org/Style/CSS/*

[4] M. Keller and M. Nussabaumer, "Cascading style sheets: a novel approach towards productive styling with today's standards," in Proc. of the International Conference on World Wide Web, pp. 1161-1162, 2005.

[5] CSS Optimizer, *http://www.cssoptimiser.com/*

[6] PCSS Compressor, *http://iceyboard.no-ip.org/projects/css_compressor*

[7] Optimiser, *http://flumpcakes.co.uk/css/optimiser/*

[8] CleanCss, *http://www.cleancss.com/*

[9] S. R. Schach, *"Object-Oriented and Classical Software Engineering,"* 5th ed., WCB/McGraw-Hill, 2002.

[10] 이은주, 박근덕, “웹 어플리케이션 재구조화를 위한 클러스터링에 사용되는 결합도 메트릭,” 한국컴퓨터정보학회논문지, 제 12권, 제 3호, 75-84쪽, 2007년 7월.

[11] B. Lee, E. Lee, and C. Wu, "Genetic Algorithm Based Restructuring of Web Applications Using Web Page Relationships and Metrics," Lecture Notes in Computer Science, Sprinter-Verlag, Vol. 4113, pp. 697-702, 2006.

[12] CSS_Architecture_Overview, *http://www.netfxharmonics.com/2007/05/CSS-Architecture-Overview*

[13] E. A. Meyer, *"Cascading Style Sheets: The Definitive Guide,"* O'Reilly Media, 2000.

저 자 소 개

정 우 성

1998 ~ 2002 : SK UBCare 연구원
2003 : 서울대학교 컴퓨터공학부(학사)
2003 ~ 2004 :
서울대학교 전기컴퓨터공학부(석사과정)
2004 ~ 현재 :
서울대학교 전기컴퓨터공학부
(석사/박사통합과정)
관심분야 : 소프트웨어 진화, 소프트웨어 마이닝, 소프트웨어 아키텍처, 웹공학, 적응형 소프트웨어

이 은 주

2005 : 서울대학교 전기컴퓨터공학부 (공학박사)
2006 ~ 현재 :
경북대학교 IT대학 컴퓨터학부 조교수
관심분야 : 웹공학, 재공학, 메트릭, 소프트웨어 유지보수