JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 21, No. 11 : 2153~2166 Nov. 2017

MVC 아키텍처 인지하는 웹 앱 재구조화

오재원[1] · 안우현[2*] · 김태공[3]

MVC Architecture-aware Restructuring of Web Apps

Jaewon Oh[1] · Woo Hyun Ahn[2*] · Taegong Kim[3]

[1]School of Computer Science and Information Engineering, The Catholic University of Korea, Bucheon 14662, Korea
[2*]School of Software, Kwangwoon University, Seoul 01897, Korea
[3]Department of Computer Engineering, Inje University, Gimhae 50834, Korea

요 약

웹 앱에는, 서로 다른 웹 페이지를 웹브라우저에 로드할 때 동일 데이터가 반복적으로 획득되고 처리되어 화면상에 표시되는 문제가 있다. 본 논문은 이 문제를 해결하기 위해 자바 웹 앱을 재구조화하는 기법을 제안하고 평가한다. 자바 웹 앱을 MVC(Model-View-Controller) 아키텍처 관점에서 동적 분석하고 컴포지트 뷰 패턴을 활용하여 중복해서 사용되는 데이터를 식별한다. 이를 토대로 페이지 요청 시 중복 데이터가 로드되지 않도록 앱을 재구조화한다. 재구조화를 통해 MVC 아키텍처에 부합하며 성능이 향상된 웹 앱을 생성한다. 이렇게 재구조화된 웹 앱은 기존 웹 앱의 응답 시간을 데스크톱 PC, 모바일 기기에서 각각 38%, 55% 개선하였다. 아울러 오픈 소스 웹 앱을 이용한 사례 연구를 통해 제안하는 기법의 적용 가능성을 보였다.

ABSTRACT

Web apps have a problem that they cause same data to be repetitively retrieved, processed, and displayed when web browsers load different web pages. To resolve the problem, this paper presents and evaluates a new method for restructuring of Java web apps. This approach dynamically analyzes Java web apps from the MVC (Model-View-Controller) architecture point of view and identifies redundant data by using the composite view pattern. Then the input apps are restructured in order not to load the redundant data when users make requests for new pages. This restructuring generates new web apps that conform to the MVC architecture and improve the performance of input web apps. The experimental results showed that when compared to legacy web apps, the restructured apps' response time was reduced on desktop PCs and mobile devices by 38% and 55%, respectively. In addition, case studies using open-source web apps showed the applicability of the proposed approach.

키워드 : 모델-뷰-컨트롤러 아키텍처, 재구조화, 단일 페이지 애플리케이션, 컴포지트 뷰 패턴, 웹 컴포넌트 상호작용

Key word : Model-View-Controller Architecture, Restructuring, Single-Page Application, Composite View Pattern, Web Component Collaboration

Received 02 August 2017, **Revised** 17 August 2017, **Accepted** 20 September 2017

* **Corresponding Author** Woo Hyun Ahn(E-mail:whahn@kw.ac.kr, Tel:+82-2-940-5760)
School of Software, Kwangwoon University, Seoul 01897, Korea

Open Access https://doi.org/10.6109/jkiice.2017.21.11.2153 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

하나의 웹 앱을 구성하는 서로 다른 웹 페이지(이하 페이지)들을 살펴보면, 페이지들이 동일한 컴포넌트를 중복 사용하는 것을 자주 발견할 수 있다[1]. 웹 앱의 메뉴, 일관된 페이지 레이아웃 같은 것이 이러한 공통 컴포넌트의 예이다. 이 경우 사용자가 페이지를 요청하면, 웹 서버(이하 서버)가 반환하는 페이지는 동일한 페이지 레이아웃을 따르고, 공통 메뉴를 탑재하며, 요청에 특화된(specific) 데이터를 포함하게 된다.

이러한 공통 데이터의 중복 사용[1]은 일관된 사용자 인터페이스(이하 UI)를 제공하는 등의 사용성(usability)을 높이는 장점이 있다[2]. 그 반면 페이지를 요청할 때마다 공통 컴포넌트가 반복적으로 네트워크로 전송되고, 웹브라우저(이하 브라우저)에서 해석되고 화면에 표시되는 문제가 있다[1, 3].

위 중복 연산 문제를 해결하기 위한 여러 재구조화 연구[3-6]가 제안되었다. 이 연구들은 기본적으로 페이지를 요청하면, 웹브라우저가 요청에 대응하는 특화 컴포넌트만을 내려 받는다. 그리고 현재 페이지에서 현재 특화 컴포넌트를 삭제하고 새 특화 컴포넌트를 삽입하도록 기존 웹 앱을 재구조화한다. 이렇게 웹 앱이 페이지 리프레시(page refresh) 없이 동작하면 공통 컴포넌트 전송 및 실행의 반복이 줄어들 수 있다.

연구 [4]에 따르면, 특화 컴포넌트만을 주고받도록 웹 앱을 재구조화하기 위해서는 우선 웹 페이지에서 특화 컴포넌트와 공통 컴포넌트를 식별할 필요가 있다. 그런데 기존 웹 앱 재구조화 연구에서 대부분 이 식별 방법을 제시하지 않는다. 연구 [4]가 이 문제를 직접 고려하지만 클라이언트 코드(HTML, CSS, JavaScript 등) 관점에서 식별하는 방법을 제안한다. 그런데 요즘 웹 앱은 서버 코드(Java, 서블릿, JSP, PHP 등)를 사용하여 비즈니스 로직을 수행하고, 이 결과를 가지고 클라이언트 코드를 동적으로 생성한다. 따라서 서버 코드 수준에서 특화 컴포넌트를 식별하기 위한 연구가 필요하다. 연구 [4] 또한 서버 코드 관점에서의 재구조화를 향후 연구로 언급하고 있다.

본 논문은 서버 코드 관점에서 특화 컴포넌트를 식별하고 이를 기반으로 웹 앱을 재구조화하는 방법을 제안한다. 연구 [7]에 따르면, MVC(Model-View-Controller) 아키텍처[8-11]가 자바 웹 앱을 개발하기 위해 널리 사용되어 왔다. 따라서 연구 [12]에서도 언급하듯이, 웹 앱의 효과적인 이해를 위해 MVC 아키텍처 관점에서의 분석이 수행될 수 있다. 그래서 본 논문은 웹 앱을 MVC 아키텍처 수준에서 모델, 뷰, 컨트롤러 컴포넌트와 이들 사이의 상호작용으로 분석하여 그래프 모델로 표현한다. 이 그래프 모델을 가지고 컴포지트 뷰(Composite view) 패턴[13-16]을 중심으로 페이지 생성 과정을 분석하여 특화 컴포넌트와 공통 컴포넌트를 식별한다. 그리고 이를 토대로 페이지 요청 시 특화 컴포넌트만이 웹브라우저에 로드되고 처리되도록 자바 웹 앱을 재구조화한다.

본 논문이 기여하는 점을 정리하면 다음과 같다. 첫째, 자바 웹 앱이 지니는 데이터와 연산의 중복 문제를 줄이는, MVC 아키텍처에 기반을 둔 재구조화 방법을 제시한다. 특히 서버 코드 관점에서 특화 컴포넌트 식별 방법을 제안한다. 둘째, 재구조화를 통해 응답 시간과 전송 데이터 양 측면에서 레거시 자바 앱의 성능을 향상시킨다. 셋째, 성장하는 모바일 환경에 맞춰, 제안하는 재구조화가 모바일 웹 앱 도메인에서 성능 향상을 가져올 수 있는 방법임을 보인다.

본 논문의 구성은 다음과 같다. II장에서는 제안하는 기법의 이해를 위해 기존 웹 앱 재구조화 연구, 웹 컴포넌트 상호작용 그래프, 컴포지트 뷰 패턴에 대해 설명한다. III장에서는 MVC 아키텍처를 인지하는 웹 앱 재구조화 방법을 제안한다. 특히 공통 컴포넌트의 중복 사용 문제를 제거하기 위해 MVC 아키텍처 관점에서 특화 컴포넌트를 식별하는 방법을 제안한다. IV장에서는 오픈 소스 웹 앱을 이용한 사례 연구를 통해 제안하는 재구조화 기법의 적용 가능성을 보인다. V장에서는 본 논문이 기여하는 점을 정리하고, 향후 연구를 제시한다.

Ⅱ. 배 경

제안하는 기법의 이해를 위해 기존 웹 앱 재구조화 연구, 웹 컴포넌트 상호작용 그래프[12], 컴포지트 뷰 패턴[13-16]에 대해 설명한다.

2.1. 웹 앱 재구조화

웹 앱의 중복 문제(필요 이상의 데이터 전송, 처리,

디스플레이)를 해결하기 위한 여러 연구[3-6]가 있다. 이 연구들은 기본적으로 페이지 리프레시(page refresh) 없이 새 페이지 요청을 처리하도록 웹 앱을 재구조화 혹은 재공학한다. [3]에 따르면, 이 기존 연구를 UI 변환 여부에 따라 두 종류로 구분할 수 있다. 첫째, UI를 재구조화하는 연구는 웹 앱의 기능을 그대로 둔 채 UI 레이아웃을 변경하거나, 새로운 위젯(widget)을 도입하거나 페이지 내비게이션 모델을 변경한다. 이런 연구로 [4, 5]가 있다. 둘째, 앱의 기능과 UI를 유지하는 전략을 취하는 연구[3, 6]가 있다.

첫 번째 방식은 주로 데스크톱 수준의 사용자 경험을 웹 앱에 도입하려고 한다. 그래서 백워드 내비게이션, 북마크 후 재접속과 같은 전통적인 웹 앱의 기능에 대한 고려가 덜하다. 반면 두 번째 방식은 전통적인 웹 앱의 사용 방식을 따르면서 성능과 비용을 줄이려는 시도로 볼 수 있다.

웹 앱 재구조화 혹은 재공학에 있어서 위 두 접근 방식은 웹 앱 요구사항과 특성에 따라 선택할 수 있는 옵션이라고 본다. 예를 들면 웹에서 데스크 톱 사용자 경험을 제공하는 것이 필요한 앱(워드 프로세서 등)을 대상으로 하면 첫 번째 접근 방식이 바람직한 선택이 될 수 있다고 본다. 두 번째 접근 방식은 연구 [3]에서 언급하듯이, 사용자가 재구조화 수행 전에 웹 앱을 사용하는 방식대로 재구조화된 앱을 사용할 수 있다는 장점이 있다. 본 연구는 두 번째 접근 방식을 취한다.

2.2. MVC 아키텍처에 기반을 둔 자바 웹 앱 재구조화

MVC 아키텍처는 앱의 효과적인 유지보수를 위해 사용할 수 있는 아키텍처이다. MVC 아키텍처는 앱을 구성하는 요소를 모델, 뷰, 컨트롤러로 나눈다. 모델은 데이터와 비즈니스 로직을 관리한다. 뷰는 데이터를 사용자에게 보여주는 프레젠테이션을 담당한다. 컨트롤러는 사용자로부터 입력을 받아 모델 데이터를 읽거나 변경하고, 적절한 뷰를 선택하는 역할을 한다. 위 세 구성요소의 상호작용을 통해, MVC 아키텍처는 변화하기 쉬운 프레젠테이션을 비즈니스 로직과 데이터와 분리하려고 한다.

레거시 자바 웹 앱은 기본적으로 JSP(JavaServer Pages), 서블릿(Servlet), 자바빈(JavaBean) 기술을 활용하였다. 이러한 앱을 MVC 아키텍처 관점에서 보면 모델은 자바빈 객체로, 뷰는 JSP 객체로, 컨트롤러는 서블릿 객체로 실현된다[3].

레거시 자바 웹 앱에서 일관된 UI를 제공하면서 코드 중복을 줄이기 위해 템플릿(Template)이 사용되어 왔다[3]. 즉 템플릿에서 메뉴, 레이아웃과 같은 공통 컴포넌트를 생성하고 관리한다. 새 페이지를 요청하면 요청에 특화된 서블릿이 실행되고, 그 수행 결과인 특화 컴포넌트를 템플릿에 포함(include)하여 완전한 페이지를 생성하고 브라우저로 전송한다. 자바빈 객체는 서블릿과 뷰 수행 중 비즈니스 데이터 조작 시 사용된다. 템플릿을 이용하는 웹 앱에서 페이지 요청을 처리하는 과정이 그림 1A에 나와 있다.

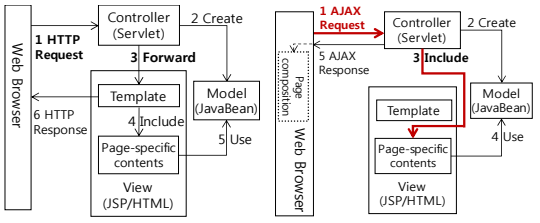
MVC 아키텍처에 기반을 두고, 멀티 페이지 웹 앱(Multi-page web app)[4]을 단일 페이지 웹 앱(Single-page web app)[4]으로 재구조화하는 연구로 [3]이 제안되었다. 멀티 페이지 웹 앱은 새 페이지를 요청하면, 브라우저에서 현재 페이지가 리프레시되고 새 페이지가 로드되는 형태로 작동하는 웹 앱이다. 반면 단일 페이지 웹 앱은 새 페이지 요청 시 현재 페이지가 리프레시되지 않고 요청 결과를 가지고 현재 페이지를 부분적으로 갱신하는 형태로 작동한다.

연구 [3]은 그림 1A에서 페이지가 서버에서 생성된다는 사실에 주목한다. 이것은 중복 데이터인, 템플릿 수행 결과가 페이지 요청마다 브라우저로 전송된다는 것을 의미한다. 이 문제를 해결하기 위해 [3]은 그림 1B와 같이 페이지가 브라우저에서 합성(composition)되도록 앱의 아키텍처를 재구조화한다. 핵심은 서버에서 응답을 생성할 때 중복되는 템플릿(공통 컴포넌트 포함)의 처리 과정을 건너뛴다는 것이다. 또한 페이지 전환이 일어나지 않도록 일반 HTTP 요청이 아니라 AJAX(Asynchronous JavaScript and XML) 요청을 이용하며 페이지 합성을 위해 요청 결과를 가지고 DOM(Document Object Model)을 갱신한다. 따라서 한 페이지 내에서 사용자 요청에 따라 상태만을 변경하는 단일 페이지 앱이 된다.

[3, 6]은 레거시 웹 변환 분야 다른 연구[4, 5]와 달리 레거시 앱이 갖는 UI를 유지하는 정책을 취한다. 즉 사용자 입장에서 레거시 앱과 동일한 UI를 통해 변환된 앱을 사용할 수 있다. 아울러 재구조화된 앱은 레거시 앱에서 사용해온 백워드 내비게이션(backward navigation)과 북마크 기능을 지원한다.

웹 앱을 효과적으로 유지보수하기 위해서는 프레젠

테이션을 담당하는 모듈과 비즈니스 로직을 담당하는 모듈이 분리되어야 한다. 그러나 연구 [3]을 통해 재구조화된 앱은 MVC 아키텍처 관점에서 분리가 되어야 할 이 두 가지 모듈이 분리가 되지 못하는 문제를 갖는다. 3장에서 이 문제를 구체적으로 설명하고, 이것을 개선한 재구조화 기법을 제안한다.



(A) Legacy app (B) Restructured app

Fig. 1 Architecture restructuring proposed by Oh et al.[3]

2.3. 웹 컴포넌트 상호작용 그래프

사용자는 하이퍼링크를 클릭하여 새 페이지를 요청한다. 이러한 요청이 발생하면, 웹 서버에서는 여러 웹 컴포넌트(JSP, 서블릿, HTML 등)가 상호작용하여 새 페이지를 생성하게 된다. 웹 앱의 동작을 이해하기 위해서 이러한 웹 컴포넌트 사이의 상호작용을 문서화할 필요가 있다[12].

연구 [12]에서는 위 상호작용을 표현하기 위해 요청 처리 경로(collaboration, request route) 모델을 제안한다. 이 모델은 그래프 기반이며 아울러 그래프에 대응하는 텍스트 모델도 제공한다. 페이지 요청 r을 처리하기 위한 요청 처리 경로는 아래 그래프 모델의 정의를 따른다. 본 논문의 이해를 위해 연구 [12]에서 제안한 모델을 축약하여 기술한다.

(정의 1). (요청 처리 경로 그래프) 페이지 요청 r이 주어질 때, r의 요청 처리 경로 그래프 $Gr(V, E, t)$는 방향 그래프이며 다음 조건을 만족한다.

- V: r의 처리 과정에서 실행되는 웹 컴포넌트(JSP, 서블릿, HTML, 이미지 등) 집합이다.
- E: V에 속하는 두 웹 컴포넌트 사이의 관계를 표현한다. V에 속하는 v_i, v_j에 대해, v_i가 v_j를 호출하면 $<v_i, v_j>$가 E에 속한다.
- t: $E \rightarrow \{Enter, Forward, Include\}$. t는 E에 속하는 각 간선의 호출 타입을 구하는 함수이다. 예를 들면, E에 속하는 $e_i(<v_j, v_k>)$에 대해, v_j가 v_k를 표준 *include* 액션으로 호출하면 $t(e_i)$=*Include*가 된다. 가능한 호출 타입은 다음과 같다.

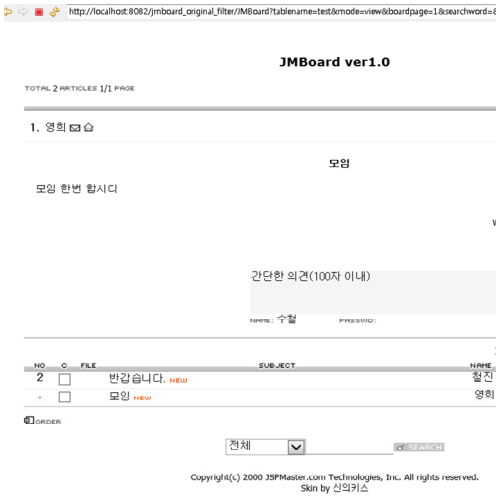
  - *Enter*: 요청 처리 시 웹 컴포넌트에서 첫 웹 컴포넌트로 가는 간선 타입이다
  - *Forward*: 표준 *forward* 액션을 호출하는 웹 컴포넌트에서 호출되는 웹 컴포넌트로 가는 간선 타입이다.
  - *Include*: 표준 *include* 액션을 호출하는 웹 컴포넌트에서 호출되는 웹 컴포넌트로 가는 간선 타입이다.

그림 2는 JMBoard(http://happycgi.com/8168)에서 게시물을 읽기 위해 하이퍼링크를 클릭할 때 일어나는 웹 앱 컴포넌트간의 상호작용과 생성된 결과 페이지를 보여준다. JMBoard는 게시판을 생성, 관리, 삭제하는 기능을 제공하는 오픈 소스 웹 앱이다. 그림 2B가 요청 처리 경로를 나타내는 그래프 모델이며 그림 2C가 그에 대응하는 텍스트 모델이다.

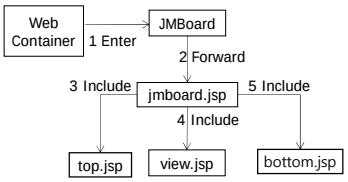
그림 2B와 그림 2C는 다음과 같은 상호작용을 나타낸다. 게시물 읽기 요청이 발생하면 이 요청이 우선 서버의 웹 컨테이너에게 전달된다. 웹 컨테이너는 이 요청에 대응하는 컨트롤러인 JMBoard를 호출한다. JMBoard가 비즈니스 로직을 수행한 후 결과 출력을 위해 뷰인 jmboard.jsp를 *forward*로 호출한다. 이 JSP는 top.jsp, view.jsp, bottom.jsp 뷰를 차례로 포함하여 결과 페이지를 생성한다. 이 결과는 웹 컨테이너를 통해 웹 브라우저에 전달된다.

텍스트 모델은 요청 처리 과정을 점진적으로 기록하여 완성한다. 즉 컴포넌트 C가 실행될 경우 C 수행 직전에 C의 수행 시작 표시(“(”)를 하고 C의 호출 타입과 C의 ID를 기록한다. 예를 들면 “Include C”는 앞선 웹 컴포넌트가 컴포넌트 C에게 *include* 요청을 호출한다는 것을 의미한다. 그리고 C와 C가 호출한 컴포넌트가 수행을 모두 마치면 C의 처리가 끝났다(“)”)는 것을 표시한다. 즉, 호출이 중첩(nesting)될 수 있으므로 괄호를 이용해 중첩 관계를 명시한다. 아울러 컴포넌트 ID로 요청 URL(Uniform Resource Locator)을 사용한다.

본 논문에서는 특화 컴포넌트(그림 2의 경우는 view.jsp)를 식별하기 위해 웹 앱의 서버 코드를 추상화하여 요청 처리 경로 모델로 표현한다. 이 모델을 다음 절에서 설명하는 컴포지트 뷰 패턴 관점에서 분석하여 특화 컴포넌트와 공통 컴포넌트를 식별한다.



(A) UI



(B) Collaboration graph model

(Enter
/**JMBoard**?tablename=test&mode=view&boardpage=1&searchword=&searchscope=&category=&no=1 (Forward
/**jmboard.jsp**?tablename=test&mode=view&boardpage=1&searchword=&searchscope=&category=&no=1 (Include
/skin/kissofgod3_gray/**top.jsp**) (Include
/skin/kissofgod3_gray/**view.jsp**) (Include
/skin/kissofgod3_gray/**bottom.jsp**)))

(C) Text model

Fig. 2 Page and collaboration of JMBoard for article reading

2.4. 컴포지트 뷰 패턴

뷰가 여러 서브뷰(subview)로 구성될 수 있다. 뉴스 기사, 날씨 정보, 주식 정보 등의 서브뷰를 모아 한꺼번에 하나의 페이지로 보여 주는 포털 사이트를 예로 들 수 있다. 이 포털 사이트는 사용자 방문 횟수를 높이기 위해 자주 페이지의 레아아웃을 변경할 수 있을 것이다. 이 경우 페이지의 레이아웃(서브뷰의 크기, 배치 등) 정보를 서브뷰 콘텐츠와 독립적으로 관리하는 것을 원할 수 있다. 이 때 사용할 수 있는 패턴이 컴포지트 뷰 패턴이다.

컴포지트 뷰 패턴은 웹 페이지를 생성할 때 비즈니스 로직의 처리를 마친 후 결과를 출력할 때 사용한다. 이 패턴은 뷰를 단일 뷰와 컴포지트 뷰로 나눈다. 단일 뷰는 내부적으로 다른 뷰를 가지지 않는 뷰이다. 이와 달리 컴포지트 뷰는 여러 개의 뷰(단일 뷰 혹은 컴포지트 뷰)를 가지는 뷰이다. 이렇게 뷰를 모듈화하고 합성 과정을 통해 최종 뷰를 생성하게 되면 뷰의 레이아웃과 서브뷰를 독립적으로 관리할 수 있다. 컴포지트 뷰 패턴은 이러한 뷰와 뷰 매니저를 구성요소로 한다. 뷰 매니저는 뷰의 콘텐츠와 레이아웃을 분리하는 역할을 한다.

컴포지트 뷰 패턴의 수행 흐름은 다음과 같다. 웹 페이지 생성 과정에서 뷰에게 결과의 출력을 요청한다. 뷰는 뷰 매니저에게 적절한 템플릿 선정을 요청한다. 뷰 매니저는 요청에 적합한 템플릿을 선택한 후 템플릿에 기술된 레이아웃을 따라 서브뷰를 합성하여 뷰/페이지를 생성한다.

[13]에 따르면, 자바 웹 앱에서 웹 페이지 생성을 위해 컴포지트 뷰 패턴이 널리 사용되어 왔다. 그래서 본 논문에서는 웹 앱의 서버 코드를 컴포지트 뷰 패턴 관점에서 분석하여 특화 컴포넌트를 식별한다. 다음 장에서 이것을 자세히 설명한다.

Ⅲ. 재구조화

본 논문은 MVC 아키텍처를 인지하는 웹 앱 재구조화 방법을 제안한다. 재구조화의 기본적인 목적은 멀티 페이지 웹 앱을 단일 페이지 웹 앱으로 변경하는 것이다. 특히 공통 컴포넌트의 중복 사용 문제를 제거하기 위해 MVC 아키텍처 관점에서 특화 컴포넌트를 식별하는 방법을 제안한다. 이번 장에서 이를 위한 재구조화 과정을 구체적으로 설명한다.

재구조화 과정이 그림 3에 나와 있다. 재구조화 과정은 크게 세 단계로 나눌 수 있다. 첫 단계는 웹 앱을 실행하여 페이지 요청을 수집하고, 각 요청별로 요청 처리 경로 그래프를 식별한다. 이를 위해 사용자 관점에서 웹 앱을 수행할 필요가 있다 (줄 번호 2). 두 번째 단계는 각 페이지 요청에 대해 요청의 행동을 표현하는 요청 처리 경로 그래프를 분석하여, 그래프를 구성하는 컴포넌트를 공통 컴포넌트와 페이지 특화 컴포넌트로

분류한다(줄 번호 3-4). 이 때 컴포지트 뷰 패턴을 활용한다. 세 번째 단계는 공통 컴포넌트의 다운로드와 처리 작업을 제거하고 특화 컴포넌트만을 가지고 현재 페이지가 갱신되도록 서버 코드와 클라이언트 코드를 변환한다(줄 번호 6).

때로는 세 번째 단계에서 언급한 변환이 어려운 경우가 있다. 본 연구의 목표는 변환의 가능 여부를 판단하는 것이 아니라, 변환이 가능한 경우 변환은 어떻게 수행되어야 하는 지에 관한 것이다. 그래서 변환의 가능 여부를 판단하기 위해서 즉, 전체 페이지 요청을 부분 페이지 갱신 요청으로 변경 가능한지 판단하는 방법으로 본 연구에서는 저자가 기존에 제안한 연구 [17]을 이용한다(줄 번호 5). 연구 [17]의 기본적인 아이디어는 기존 전체 페이지 요청의 결과와 부분 페이지 갱신 요청의 결과를 MVC 아키텍처 관점에서 비교 분석하여, 두 결과 사이에 불일치가 발생하지 않으면 재구조화를 수행한다는 것이다. 즉, 기존 전체 페이지 요청과 비교했을 때 요청을 변환하게 되면 다른 모델, 다른 뷰가 생성되거나 다른 컨트롤러로 요청이 전달되는 등의 불일치가 발생하면 변환하지 않는다.

그림 3에서 언급한 단계 중 핵심 단계인 두세 번째 단계를 다음 두 개의 절에서 자세히 설명한다.

```
1: procedure restructure(App app)
2: collect collaboration diagrams for app's every page request
3: for each page request r
4:   findPageSpecificComponent(r)
5:   if (r's transformation does NOT induce an inconsistent
state from the perspective of MVC architecture) then
6:     transform(r)
7:   end if
8: end for
9: end procedure
```

Fig. 3 MVC architecture-aware restructuring

3.1. 특화 컴포넌트 식별

우선 웹 앱의 서버 코드를 요청 처리 경로 그래프 모델로 추상화한다. 이 추상화를 위해 2.3절에서 소개한, 저자의 기존 연구[12]를 이용할 수 있다. 본 논문의 목표는 웹 앱의 행동을 그래프 모델로 표현하는 것이 아니라, 그래프 모델에 기반을 두고 재구조화를 수행하는 방법에 관한 것이다. 그러고 나서 이 모델을 컴포지트 뷰 패턴 관점에서 분석하여 특화 컴포넌트와 공통 컴포넌트를 식별한다.

그림 4에 이러한 식별 알고리즘이 나와 있다. 현재 고려하는 페이지 요청 r의 요청 처리 경로 그래프에 등장하는 웹 컴포넌트를 컴포지트 뷰 패턴에 참여하는 객체 타입으로 분류한다(줄 번호 4). 이러한 객체 타입에는 템플릿, 뷰 매니저, 서브뷰, 복합 뷰가 있다. 예를 들면 JMBoard에서는 그림 2, 그림 5, 그림 6과 같은 요청 처리 경로 그래프를 추출할 수 있다. 게시글 읽기 요청, 게시글 수정 UI 요청, 게시글 수정 요청에 대해서 UI와 요청 처리 경로 모델(그래프 모델과 텍스트 모델)이 세 그림에 각각 나와 있다.

이 요청 처리 경로 모델에 계속적으로 등장하는 컴포넌트 중에서 jmboard.jsp가 템플릿 역할을 한다. 왜냐하면 jmboard.jsp는 최종 페이지 생성을 위한 페이지 레이아웃 정보를 가지며 페이지를 구성하는 여러 서브뷰가 페이지에 포함되도록 표준 *include* 태그를 사용하고 있기 때문이다. 한편 이 예제에서 뷰 매니저는 자바 웹 컨테이너로 볼 수 있으며, 표준 태그 뷰 관리 전략 [13]을 사용하며 표준 *include* 태그를 사용하여 서브뷰를 복합 뷰에 포함시킨다. 서브뷰로는 헤더(top.jsp), 푸터(bottom.jsp), 페이지 특화 뷰가 식별된다. 페이지 특화 뷰는 레이아웃 관점에서 헤더와 푸터 사이에 위치하는 컴포넌트로 페이지 요청에 특화된 콘텐츠를 보여준다. 예를 들면 그림 6의 경우 list.jsp가 특화 뷰이다. 복합 뷰로는 템플릿을 따라 서브뷰를 포함하여 만들어진, 사용자에게 보이는 최종 페이지(예를 들면 그림 6A)가 있다.

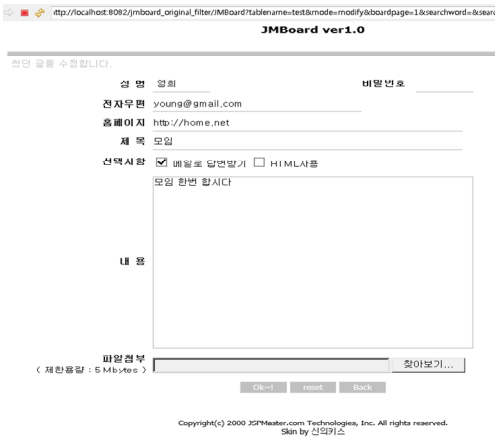
```
1: procedure findPageSpecificComponent(Request r)
2: // to identify and classify subviews composing a whole
view of request r into two categories: common and
page-specific subviews
3: // let svp_r^i denote whether a subview i of request r is a
page-specific (true) or not (false)
4: classify web components of r's collaboration diagram into
templates, view managers, subviews, or composite views
5: for each subview sv_r^i of r
6:   if (sv_r^i∈cd_j for all collaboration diagram cd_j) then
7:     svp_r^i ← false
8:   else
9:     svp_r^i ← true
10:  end if
11: end for
12: end procedure
```

Fig. 4 Finding page-specific and common components

이러한 최종 페이지를 구성하는 서브뷰는 공통 컴포넌트나 특화 컴포넌트로 나눌 수 있다. 여러 페이지에 공통적으로 배치되는 컴포넌트가 공통 컴포넌트인데 (줄 번호 6-7), 이 JMBoard 예제에서는 헤더와 푸터가 이것으로 분류될 수 있다. 새 페이지 요청에 대응하여 해당 페이지에서만 배치되는 것이 특화 컴포넌트인데 (그림 4 줄 번호 6, 9), 이 예제에서는 view.jsp, write.jsp, list.jsp가 이것으로 분류될 수 있다.

3.2. 변환

지금까지 페이지 요청에 대응하는 요청 처리 경로 그래프를 구하고 그래프를 구성하는 웹 컴포넌트의 역할을 컴포지트 뷰 관점에서 파악하였다.



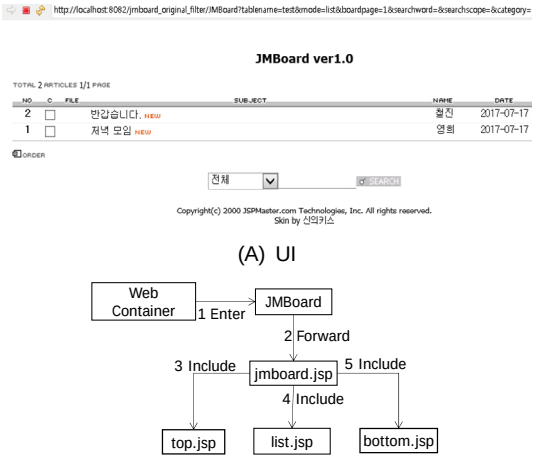
(A) UI

(B) Collaboration graph model

(Enter
/**JMBoard**?tablename=test&mode=modify&boardpage=1&searchword=&searchscope=&category=&no=1 (Forward
/**jmboard.jsp**?tablename=test&mode=modify&boardpage=1&searchword=&searchscope=&category=&no=1 (Include
/skin/kissofgod3_gray/**top.jsp**) (Include
/skin/kissofgod3_gray/**write.jsp**) (Include
/skin/kissofgod3_gray/**bottom.jsp**)))

(C) Text model

Fig. 5 Data entry page for article update on JMBoard and its collaboration



(A) UI

(B) Collaboration graph model

(Enter
/**JMBoard**?tablename=test&mode=list&boardpage=1&searchword=&searchscope=&category= (Forward
/**jmboard.jsp**?tablename=test&mode=list&boardpage=1&searchword=&searchscope=&category= (Include
/skin/kissofgod3_gray/**top.jsp**) (Include
/skin/kissofgod3_gray/**list.jsp**) (Include
/skin/kissofgod3_gray/**bottom.jsp**)))

(C) Text model

Fig. 6 Result page of article update on JMBoard and its collaboration

이제 그림 7에 특화 컴포넌트만을 주고받도록 웹 앱을 변환하는 알고리즘이 나와 있다. 세 가지 경우로 나누어서 변환이 수행된다. 첫째, 특화 컴포넌트와 공통 컴포넌트 모두를 템플릿에서 포함하는 경우이다. 이 경우는 일반적인 컴포지트 뷰 패턴을 따르는 형태이며, 템플릿을 수정하여 템플릿에 특화 컴포넌트만 포함되도록 변환한다(그림 7 줄 번호 2-4).

그림 8A에 요청 *ri*와 *rj*를 변환하는 예제가 나와 있다. 이 그림에서 *T*가 템플릿이다. *Co1*, *Co2*가 공통 컴포넌트이다. *Spi*와 *Spj*는 각 페이지 요청에 알맞은 특화 컴포넌트이다. *Ci*, *Cj*는 요청에 적합한 컨트롤러를 의미한다. 이 경우 변환을 하면 그림 8B와 같다. 공통 컴포넌트가 포함되지 않도록 템플릿을 수정하면 된다.

변환 후 페이지 요청은 다음과 같이 웹 서버에서 처리된다(그림 8B). 페이지 요청이 발생하면 변환되기 전처럼 우선 관계된 컨트롤러가 수행되어 비즈니스 로직을 수행한다. 그 후 결과의 출력을 위해 템플릿으로 요청 처리가 포워드된다.

```
1: procedure transform(Request r)
2: if (r's page-specific subview and r's common subviews are all included by r's template) then
3:     the template is modified such that the common subviews are removed from r's collaboration diagram
4:   modify r such that r becomes a partial-page request
5: else if (r's page-specific subview is included by r's common subview) then
6:    a template is constructed such that the layout info is extracted from the common view and is moved to the template
7:   the remainer of the common view is partitioned into two parts: the part executed before the page-specific subview and the part executed after the page-specific subview
8:   Do step 3-4
9: else if (r's page-specific view includes r's common views) then
10:    a template is constructed such that the layout info is extracted from the page-specific view and is moved to the template
11:    the remainer of the page-specific view is positioned between the common views
12:   Do step 3-4
13: end if
14: end procedure
```

Fig. 7 Transforming a full-page request into a partial-page request with page-specific content

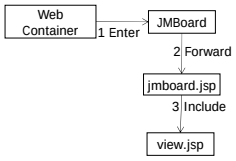
재구조화된 템플릿에서는 재구조화 전과 달리 특화 컴포넌트만을 포함하여 결과를 브라우저로 반환한다. 아울러 변환 전에는 웹브라우저에서 반환 결과를 보여주기 위해 현재 페이지를 리프레시했지만, 변환 후에는 반환 결과를 가지고 현재 페이지를 부분 업데이트한다.

JMBoard에서 게시글 읽기 요청(그림 2)을 변환할 때 변환 후의 경로 요청 모델이 그림 9에 나와 있다. 템플릿 역할을 하는 jmboard.jsp를 수정하여 특화 컴포넌트(view.jsp)만 로드되도록 한다. 다른 요청(그림 5, 그림 6)도 같은 방식으로 변환할 수 있다.

Request ri: (En Ci (FW T (In Co1) (In Spi) (In Co2)))
Request rj: (En Cj (FW T (In Co1) (In Spj) (In Co2)))
(A) Collaboration models of original requests

Request ri: (En Ci (FW T' (In Spi)))
Request rj: (En Cj (FW T' (In Spj)))
(B) Collaboration models of restructured requests
En, FW, and In mean Enter, Forward, and Include, respectively.

Fig. 8 Request transformation when all subviews of an original request are included by a template



(A) Collaboration graph model

(Enter /**JMBoard**?tablename=test&mode=view&boardpage=1&searchword=&searchscope=&category=&no=1 (Forward /**jmboard.jsp**?tablename=test&mode=view&boardpage=1&searchword=&searchscope=&category=&no=1 (Include /skin/kissofgod3_gray/**view.jsp**)

(B) Text model

Fig. 9 Request transformation result in case of article reading on JMBoard

두 번째와 세 번째는 템플릿이 독립적으로 존재하지 않은 경우이다. 그래서 두 번째는 공통 컴포넌트가 템플릿 역할까지 수행하는 경우이다. 이 경우의 요청 처리 경로 모델과 변환 방식이 그림 10에 나와 있다. 컴포넌트 *Co*가 템플릿과 공통 컴포넌트 역할을 겸하고 있다(그림 10A).

본 논문에서는 연구 [3, 17]에서처럼 페이지 특화 컴포넌트가 페이지의 한 곳에 공간적으로 지역화되어 있다고 가정한다. 즉, <*span*> 이나 <*div*> 태그 한 개로 특화 컴포넌트를 묶을 수 있다. 그래서 이 예제에서도 특화 컴포넌트가 요청별로 각 *Spi*, *Spj*로 한 곳에 지역화되어 있는 것을 볼 수 있다.

이 경우 변환을 두 단계로 나누어 수행한다(그림 7 줄 번호 5-8). 첫 단계에서는 공통 컴포넌트에서 레이아웃 정보를 추출하여 템플릿을 생성한다(그림 7 줄 번호 6). 특화 컴포넌트 전과 후로 수행하던 공통 작업을 새로운 두 개의 공통 컴포넌트에 할당한다(그림 7 줄 번호 7, 그림 10B 참고).

그러면 이 요청 처리 경로 모델의 모습이 그림 8A 형태가 된다. 이제 그림 8에서 제안하는 방법대로 변환을 수행하면 된다(그림 10C 참고).

셋째, 특화 컴포넌트가 템플릿 역할까지 수행하는 경우이다. 이 경우의 요청 처리 경로 모델과 변환 방식이 그림 11에 나와 있다. 각 요청별로 컴포넌트 *Spi*, *Spj*가 템플릿과 특화 컴포넌트 역할을 겸하고 있다.

Request ri: (En Ci (FW Co (In Spi)))
Request rj: (En Cj (FW Co (In Spj)))
(A) Collaboration models of original requests

Request ri: (En Ci (FW T (In Co3) (In Spi) (In Co4)))
Request rj: (En Cj (FW T (In Co3) (In Spj) (In Co4)))
(B) Collaboration models of requests in intermediate steps

Request ri: (En Ci (FW T' (In Spi)))
Request rj: (En Cj (FW T' (In Spj)))
(C) Collaboration models of restructured requests

Fig. 10 Request transformation when a page-specifc subview of an original request is included by a common subview

Request ri: (En Ci (FW Spi (In Co5) (In Co6)))
Request rj: (En Cj (FW Spj (In Co5) (In Co6)))
(A) Collaboration models of original requests

Request ri: (En Ci (FW T (In Co5) (In Sp'i) (In Co6)))
Request rj: (En Cj (FW T (In Co5) (In Sp'j) (In Co6)))
(B) Collaboration models of requests in intermediate steps

Request ri: (En Ci (FW T' (In Sp'i)))
Request rj: (En Cj (FW T' (In Sp'j)))
(C) Collaboration models of restructured requests

Fig. 11 Request transformation when a page-specific view of an original request includes common subviews

이 경우에도 변환을 두 단계로 나누어 수행한다(그림 7 줄 번호 9-12). 첫 단계에서는 특화 컴포넌트에서 레이아웃 정보를 추출하여 템플릿을 생성한다(그림 7 줄 번호 10). 공통 컴포넌트 사이에서 수행하는 특화 작업을 위해 새로운 특화 컴포넌트를 마련한다(그림 7 줄 번호 11, 그림 11B 참고). 그러면 이 요청 처리 경로 모델의 모습이 그림 8A와 같아진다. 그림 8에서 제안하는 방법대로 변환을 수행한다(그림 11C 참고).

3.3. 소프트웨어 아키텍처 관점에서의 재구조화 이슈

첫째, 요청 처리 경로 그래프를 구성하는 웹 컴포넌트 사이의 상호작용의 변환에 대한 이슈가 있다. 기존 MVC 아키텍처 기반 웹 앱 재구조화 연구 [3]은 그림 1B의 단계 3에서 보듯이 특화 컴포넌트만을 반환하기 위해, 사용할 수 있는 커넥터(connector)로 표준 *include* 태그를 이용한다. 그런데 이 *include* 태그를 사용하면 호출 전후 자유롭게 HTML 코드 생성이 가능하다. 즉 컨트롤러가 단계 3 이전에 프레젠테이션 코드를 생성할 수 있게 된다. 이것은 MVC 아키텍처의 프레젠테이션과 로직의 분리 원리를 어긴다.

본 논문에서는 위 문제를 해결하기 위해 커넥터로 표준 *forward* 태그 사용을 제안한다. 이 커넥터는 *forward* 이후에만 프레젠테이션 코드를 생성할 수 있어, 로직과 프레젠테이션의 분리가 가능하다. 그림 1A가 입력으로 주어질 때 재구조화된 앱의 작동 모습이 그림 12A에 나와 있다.

한편 페이지 전환 없이 특화 컴포넌트만 전송 받고 웹 앱의 상태를 변화하려면 사전에 서버로부터 공통 컴포넌트를 다운로드해야 한다. 이렇게 전체 페이지 다운로드가 필요한 경우는 처음으로 웹 앱에 접속할 때, 웹 앱의 특정 상태를 북마크한 후 재접속할 때, 사용자가 현재 페이지를 리프레시할 때 등이다. 이 경우 [3]은 레거시 앱과 같이 서버에서 페이지를 합성한 후 브라우저로 전달한다. [17]은 레거시 앱에 존재하는 병렬성을 이용하여 [3]의 전체 페이지 다운로드 성능을 향상하는 방법을 제안한다. 본 논문은 [17]에서 제안한 방법을 사용한다. 즉, 우리는 페이지가 공통 컴포넌트와 특화 컴포넌트로 구성되며 두 컴포넌트가 서로 무관하다는 점을 이용한다. 그래서 전체 페이지 다운로드 시 [3]과 달리 두 컴포넌트를 각기 다른 요청을 통해 다운로드하고 브라우저에서 최종 페이지를 합성한다(그림 12B 참고). 즉 1단계에서 HTTP 요청을 통해 공통 컴포넌트를 로드하고 3단계에서 AJAX 요청을 통해 특화 컴포넌트를 로드한다.

아키텍처 관점에서 주목할 만한 또 다른 이슈로 백워드 내비게이션 지원 정책이 있다. 여러 페이지로 구성된 웹 앱에서 사용자 요청에 따라 페이지 $p_1, p_2, ..., p_n$을 차례로 방문했다고 하자. 현재 페이지 p_n에서 백 버튼을 눌러 백워드 내비게이션을 수행하면, 이전 페이지인 p_{n-1}로 되돌아간다. 이와 달리 단일 페이지 웹 앱 *app1*에서는 사용자 요청에 따라 하나의 페이지 내에서 상태가 변하게 된다. 사용자 요청에 따라 상태 $s_1, s_2, ..., s_n$을 차례로 방문했다고 하자. 현재 상태 s_n에서 백 버튼을 눌러 백워드 내비게이션을 수행하면, 이전 상태인 s_{n-1}

이 아니라 *app1*을 방문하기 직전 페이지로 되돌아간다. 그렇지만 전통적인, 여러 페이지로 구성된 웹 앱에 익숙한 사용자는 s_{n-1} 상태로 되돌아가는 것을 원할 수 있다. 이럴 경우, 웹 앱은 자체적으로 백워드 내비게이션 시 s_{n-1}, s_{n-2}, ..., s_1을 방문하는 방법을 제공해야 한다.

이러한 이전 상태의 방문을 위해, 두 가지 방식을 생각할 수 있다. 이전 상태를 그대로 저장해 두고 재방문 시 이 상태를 사용할 수 있다. 혹은 이전 상태를 생성하는 URL을 저장해 두고 재방문 시 이 URL로 요청하여 그 결과를 사용할 수 있다. 첫 방식은 속도 면에서 이점이 있으나 서버 데이터와의 일관성 측면에서 문제가 발생할 수 있다. 어느 방식을 선택하느냐는 웹 앱의 특성에 의존한다고 본다. 특화 컴포넌트의 변경이 잦고 항상 최신 데이터를 사용자가 원하는 경우 두 번째 접근 방식을 취할 수 있다. 그렇지 않은 경우 첫 번째 접근 방식을 선택할 수 있을 것이다. 본 논문이 제안하는 재구조화 방법은 웹 앱의 특성에 따라 한 가지 방법을 선택할 수 있다. 이와 달리 [3]의 재구조화 결과로 생성되는 단일 페이지 앱은 이전 상태 방문을 위해 첫 번째 방식을 취하고 있지만, 이러한 선택에 대한 근거를 기술하고 있지 않다.

3.4. 구현

소스코드 수준에서 재구조화 예가 그림 13에 나와 있다. 그림 13A에서 사용자가 책 추가 하이퍼링크를 클릭하면(줄 번호 1), 관련 컨트롤러가 책을 추가하는 로직을 수행한다(줄 번호 3). 그 후 컨트롤러는 결과 페이지를 생성하기 위해 템플릿(*Template.jsp*)을 *forward* 로 호출한다(줄 번호 4). 템플릿은 페이지 헤더와 메뉴와 같은 공통 컴포넌트를 포함하고(줄 번호 5), 요청에 특화된 컴포넌트를 포함한다(줄 번호 6). 마지막으로 푸터와 같은 공통 컴포넌트를 포함하고(줄 번호 7), 완성된 페이지를 웹브라우저로 반환한다.

그림 13A를 재구화한 결과가 그림 13B에 나와 있다. 요청에서 *href* 속성을 삭제하고 *onclick* 이벤트 핸들러를 통해 AJAX 요청을 한다(그림 13A, B 줄 번호 1). 컨트롤러에서 템플릿을 건너뛰고 특화 컴포넌트로 요청을 *forward*한다(그림 13A, B 줄 번호 4). 특화 컴포넌트가 AJAX 응답을 통해 도착하면 현재 페이지에 특화 컴포넌트를 삽입하도록 템플릿 코드를 수정한다(그림 13B 줄 번호 6, 9).

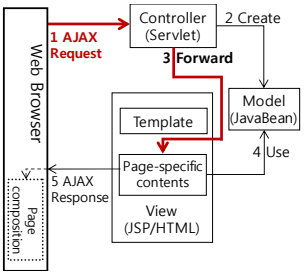
Ⅳ. 평 가

이번 장에서는 오픈 소스 웹 앱을 이용한 사례 연구를 통해 제안하는, 특화 컴포넌트 식별 기법과 이 기법에 기반을 둔 재구조화 방법의 적용 가능성을 보인다. 또한 성능 실험을 통해 재구조화된 웹 앱이 기존 웹 앱의 응답 시간을 데스크톱 PC, 모바일 기기에서 각각 38%, 55% 개선함을 보인다.

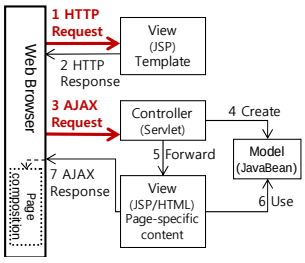
4.1. 적용 가능성 검증

적용 가능성을 검증하기 위해 첫 번째로 오픈 소스 웹 앱인 JMBoard를 실험 대상으로 삼았다. 이 웹 앱을 선택한 이유는 유명 오픈 소스 저장소에 코드가 공개되어 있고, 서블릿, JSP, 표준 액션 태그, HTML, JavaScript 등으로 구성되어 있어 Java 웹 앱의 전형적인 구성을 따르기 때문이다[12].

3장에서 재구조화를 설명하기 위해 사용한 JMBoard는 게시판을 생성하고 관리하는 것을 지원하는 웹 앱이다. 이 앱은 템플릿이 공통 컴포넌트와 특화 컴포넌트를 포함하는 형태이다. 그래서 그림 7의 줄 번호 2-4를 따라서 변환하였다. 재구조화 결과로 생성된 단일 페이지 웹 앱은 페이지 부분 업데이트, 백워드 내비게이션



(A) Client-side page composition



(B) On the first access to restructured app

Fig. 12 Proposed architecture restructuring

을 지원하기 위해 PJAX 라이브러리[18]를 사용하였다. 3장에서 제안하는 웹 앱 재구조화 절차를 따라 단일 페이지 웹 앱으로 구조화할 수 있었다.

View	1: 2:	<a **href="AddBook"**>Add book <img src="friends.png" **onclick="location.href='SearchFriend'"**>

Controller	3:
	4:
		// execute business logic & create JavaBean objects
		req.getRequestDispatcher(**"Template.jsp?Body=View2"**).**forward**(req, resp);

Template	5:
	6:
	7:
		// common component such as header, menu
		<jsp:include page="${param.Body}" /> // server-side composition
		// common component such as footer

(A) Legacy app

View	1: 2:	<a **onclick="doGetByAjax ('AddBook');"**>Add book <img src="friends.png" **onclick="doGetByAjax('SearchFriend');"**>

Controller	3:
	4:
		// execute business logic & create JavaBean objects
		req.getRequestDispatcher(**"View2"**).**forward**(req, resp);

Template	5:
	6:
	7:
	8:
	9:
		// common component such as header, menu
		** ** // client-side composition
		// common component such as footer
		// client-side composition in event handler of AJAX request
		document.getElementById("container").innerHTML= xhr.responseText;

(B) Restructured app

Fig. 13 Restructuring user requests at the code level

적용 가능성을 검증하기 위해 두 번째로 기존 웹 앱 재구조화 연구 [3]에서 소개된 온라인 출판사 앱을 실험 대상으로 삼았다. 이 앱은 공통 컴포넌트가 템플릿의 역할을 겸하여 특화 컴포넌트가 공통 컴포넌트로 포함되는 형태이다. 그래서 그림 7의 줄 번호 5-8를 따라서 변환되었다. 재구조화 결과로 생성된 단일 페이지 웹 앱은 페이지 부분 업데이트, 백워드 내비게이션 지원을 위해 AJAX와 HTML5 history API[19]를 사용하였다. 3장에서 제안하는 웹 앱 재구조화를 따라 단일 페이지 웹 앱으로 재구조화할 수 있었다. 또한 다음 절에서 온라인 출판사 원본과 재구조화 버전을 대상으로 수행한 성능 실험의 결과를 소개한다.

이번 절에서 온라인 출판사와 온라인 게시판 앱을 대상으로 재구조화를 수행한 결과를 설명하였다. 제안하는 재구조화가 다른 웹 앱에도 적용 가능하다고 보며 근거는 다음과 같다. 첫째, JSP 모델 2[9]는 MVC 아키텍처의 일종으로 산업계에서 자바 웹 앱 개발을 위해 널리 사용되어 왔다[7]. 본 논문은 이러한 JSP 모델 2를 따르는 레거시 웹 앱을 재구조화의 대상으로 삼는다. 둘째, 4장에서 재구조화를 적용한 온라인 출판사 웹 앱은 전형적인 JSP model 2 웹 앱이며 아울러 이전 연구 [7, 20]에서 벤치마크로 사용한 온라인 서점 웹 앱의 축소판이다. 셋째, 온라인 게시판 웹 앱 또한 전형적인 JSP model 2 웹 앱이며 아울러 이전 웹 앱 재구조화/역공학 연구[1, 12]에서 사례 연구로 사용되었다.

4.2. 성능 실험 결과와 분석

재구조화된 앱이 레거시 자바 앱의 성능을 향상시킨다는 것을 보이기 위해 [3]에서 사용한 온라인 출판사 앱(이하 *Tapp*)을 가지고 실험한다. *Tapp*의 재구조화 결과를 Sapp라 하자. 응답 시간과 네트워크 전송 데이터 양을 측정한다. 3.3GHz Intel i5-2500 프로세서, Windows 7 32비트 운영 체제를 갖는 두 PC를 서버와 클라이언트로 사용한다. 4GB DDR3 램, Tomcat 7.0 웹 서버, Cubrid 2008 R4.1 데이터베이스를 서버 PC에 설치하고 클라이언트 PC는 2GB DDR3 램, Chrome 36.0.1985.103m 브라우저를 사용한다.

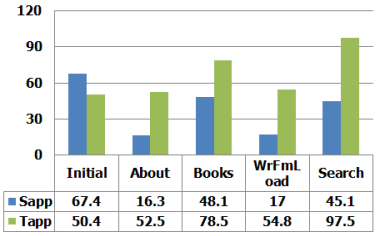
브라우저 캐싱이 웹 앱의 성능에 영향을 미친다. 그래서 두 가지 경우로 나누어 실험을 진행한다. 페이지를 처음 접근하여 캐시 미스(miss)가 발생하는 경우와 페이지를 두 번째 접근하여 캐시 히트(hit)가 발생하는 경우이다.

그림 14는 페이지를 처음 접근할 때 측정된 성능을 보여준다. 이 중 *Initial* 페이지는 특화 컴포넌트 없이 공통 컴포넌트(템플릿 포함)만 지닌 페이지이다. 이 페이지를 실험 앱 페이지들 중 가장 먼저 방문하여 공통 컴포넌트의 다운로드 및 처리 성능을 파악한다. 그 결과 AJAX 요청 처리를 위한 자바스크립트의 다운로드와 실행으로 *Sapp*가 *Tapp*에 비해 응답 시간이 느리다. 그렇지만 공통 컴포넌트 로드 이후 다른 페이지를 처음 방문할 때는 브라우저에서 페이지 합성이 이루어지는 *Sapp*가 *Tapp*에 비해 빠르다(그림 14A 참고). 데이터 측면에서는 *Sapp*는 첫 페이지인 *Intial* 페이지 이후 특화 컴포넌트만 다운로드 처리하므로 서버에서 페이지가 합성되어 다운로드되는 *Tapp*에 비해 네트워크 전송 데이터 양이 적다(그림 14B 참고). 실험에 사용된 페이지들에 포함된 특화 컴포넌트는 다양성을 위해 정적 컴포넌트와 데이터베이스 검색 후 HTML를 생성하는 동적 컴포넌트 등을 포함한다. 구체적으로 *About*은 회사 소개, *Books*는 책 목록 다운로드, *WrFmLoad*는 책 리뷰 작성 위한 폼 다운로드, *Search*는 키워드 통한 책 검색

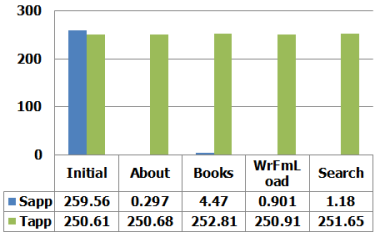
페이지이다.

그림 15는 페이지 두 번째 방문 시 성능을 보여준다. 주목할 점은 페이지를 구성하는 그림, HTML, CSS와 같은 정적 컴포넌트가 이미 브라우저에 캐싱되어 있어 첫 실험에 비해 전송 데이터 양과 응답 시간이 줄어든 것을 볼 수 있다. *Sapp*가 전체적으로 *Tapp*의 성능을 향상시킨 것을 볼 수 있다. 특히 *About* 페이지의 경우 특화 컴포넌트가 정적 컴포넌트이어서 *Sapp*는 다운로드 작업이 없어 *Tapp*에 비해 상당히 빠르다.

또 다른 기여도인, 재구조화를 통해 모바일 기기에서 더 많은 성능 향상이 가능함을 보이기 위해 모바일 브라우저에서 실험한다. 실험용 스마트 폰은 1.2GHz 프로세서, 1GB 램, 안드로이드 4.1.2 젤리 빈 운영 체제, chrome 36.0.1985.131 브라우저를 내장한 SHW-M250S이다. 그림 16은 그림 14와 그림 15의 경우에 실험한 결과이다. 그림 16을 통해 제안하는 기법을 이용하면 모바일 기기에서 더욱 향상된 성능을 이끌어 낼 수 있음을 알 수 있다. 이런 결과를 가져온 이유 중 하나는 페이지 전환 작업이 페이지 언로드와 로드를 위해 CPU 시간을 많이 사용하기 때문이라 볼 수 있다. 전송 데이터 결과는 데스크톱 실험과 동일하다. 그래서 스마트 폰 사용자가 데이터 사용량에 따라 요금을 낼 경우, *Sapp*를 사용하면 비용을 절약할 수 있다[17].



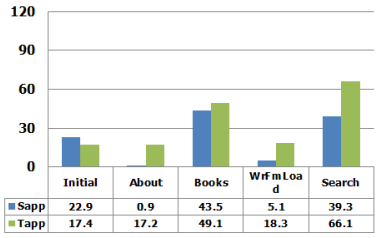
(A) Response time (ms)



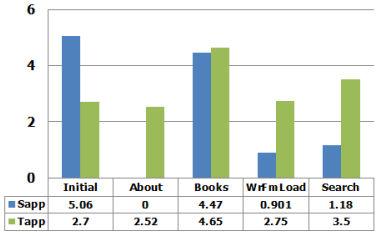
(B) Transferred data (KB)

Fig. 14 Performance of restructured app on the first access to each page

그림 14, 15, 16을 통해 재구조화된 웹 앱이 기존 웹 앱의 응답 시간을 데스크톱 PC, 모바일 기기에서 평균적으로 각각 38%, 55% 개선함을 알 수 있다.

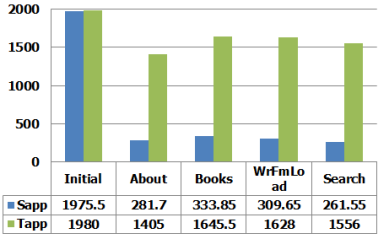


(A) Response time (ms)

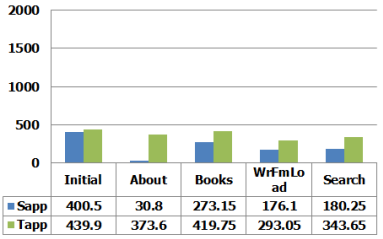


(B) Transferred data (KB)

Fig. 15 Performance of restructured app on the second access to each page



(A) On the first access



(B) On the second access

Fig. 16 Response time (ms) of restructured app on mobile device

Ⅴ. 결 론

레거시 자바 웹 앱이 지닌, 데이터와 연산의 중복 문제를 줄이기 위한 재구조화 방법을 제안한다. 이를 위해 우선 자바 웹 앱의 서버 코드를 MVC 아키텍처 관점에서 분석하여 요청 처리 경로 그래프 모델로 표현한다. 이 그래프 모델을 가지고 컴포지트 뷰 패턴을 중심으로 페이지 생성 과정을 분석하여 특화 컴포넌트와 공통 컴포넌트를 분리한다. 그리고 이를 토대로 페이지 요청 시 특화 컴포넌트만이 로드되고 처리되도록 자바 웹 앱을 재구조화한다.

본 논문이 기여하는 점은 다음과 같다. 첫째, 이 방법은 기존 재구조화 연구 [3]이 갖는 MVC 원리 위배 문제를 해결한다. 즉, 연구 [3]을 통해 재구조화된 앱은 MVC 아키텍처 관점에서 분리가 되어야 할 프레젠테이션 모듈과 로직 모듈이 분리가 되지 못하는 문제를 갖는다. 본 논문은 이 문제를 발견하고 해결한다. 둘째, 기존 웹 앱 재구조화에서 고려가 덜한 특화 컴포넌트 식별 문제에 대한 해결책을 제시한다. 셋째, 재구조화를 통해 응답 시간과 전송 데이터 양 측면에서 레거시 앱의 성능을 향상시킨다. 넷째, 성장하는 모바일 환경에 맞춰, 제안하는 재구조화가 모바일 웹 앱 도메인에서 성능 향상을 가져올 수 있는 방법임을 보였다.

향후 연구로는 지원 도구 개발을 꼽을 수 있다. 이를 위해 [3]에서 제시한 도구를 수정하여 개선할 수 있다고 본다.

ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2017R1D1A1B030 29374 and No. 2011-0013781). The present Research has been conducted by the Research Grant of Kwangwoon University in 2017.

REFERENCES

[1] J. Oh, W. Ahn, and T. Kim, “Web app restructuring based on shadow DOMs to improve maintainability,” in *Proceedings of the 8th IEEE International Conference on Software Engineering and Service Science*, Beijing, pp. 118-122, 2017.

[2] C. Kim and K. Shim, “TEXT: automatic template extraction from heterogeneous web pages,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 612-626, Aug. 2011.

[3] J. Oh, W. Ahn, S. Jeong, J. Lim, and T. Kim, “Automated transformation of template-based web applications into single-page applications,” in *Proceedings of the IEEE 37th Annual Computer Software and Applications Conference*, Kyoto, pp. 292-302, 2013.

[4] A. Mesbah and A. Van Deursen, “Migrating multi-page web applications to single-page AJAX interfaces,” in *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, Amsterdam, pp. 181-190, 2007.

[5] A. Garrido, G. Rossi, and D. Distante, “Model refactoring in web applications,” in *Proceedings of the 9th IEEE International Workshop on Web Site Evolution*, Paris, pp. 89-96, 2007.

[6] J. Chu and T. Dean, “Automated migration of list based JSP web pages to AJAX,” in *Proceedings of the 8th IEEE International Working Conference on Source Code Analysis and Manipulation*, Beijing, pp. 217-226, 2008.

[7] B. Kurniawan and J. Xue, “A comparative study of web application design models using the java technologies,” in *Advanced Web Technologies And Applications*, 1st ed. Berlin, Springer Berlin Heidelberg, pp. 711-721, 2004.

[8] S. Caballé, J. A. Ortega, J. M. Camps, L. Barolli, E. Kulla, and E. Spaho, “A presentation framework to simplify the development of java EE application thin clients,” in *Proceedings of the 2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems*, Birmingham, pp. 421-426, 2014.

[9] G. Seshadri. Understanding javaserver pages model 2 architecture [Internet]. Available: http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html.

[10] I. H. Sarker and K. Apu, “MVC architecture driven design and implementation of java framework for developing desktop application,” *International Journal of Hybrid*

Information Technology, vol. 7, no. 5, pp. 317-322, Sept. 2014.

[11] D. P. Pop and A. Altar, "Designing an MVC model for rapid web application development," *Procedia Engineering*, vol. 69, pp. 1172-1179, Mar. 2014.

[12] J. Oh, W. Ahn, and T. Kim, "Automatic extraction of component collaboration in java web applications by using servlet filters and wrappers," *KIPS Transactions on Software and Data Engineering*, vol. 6, no. 7, pp. 329-336, Aug. 2017.

[13] D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*, 2nd ed. Palo Alto, Sun Microsystems, Inc., 2003.

[14] Z. Mushtaq, G. Rasool, and B. Shahzad, "Extendable and customizable features for the detection of JEA/J2EE patterns," in *Proceedings of the 2016 International Conference on Open Source Systems & Technologies*, Lahore, pp. 151-155, 2016.

[15] Z. Mushtaq and G. Rasool, "Multilingual source code analysis: state of the art and challenges," in *Proceedings of the 2015 International Conference on Open Source Systems & Technologies*, Lahore, pp. 170-175, 2015.

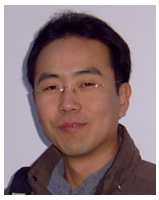
[16] N. J. Bien and T. D. Thu, "Graphical user interface variability architecture pattern," In *Proceedings of the Sixth International Symposium on Information and Communication Technology*, Hue City, pp. 304-311, 2015.

[17] J. Oh, W. Ahn, and T. Kim. "MVC architecture driven restructuring to achieve client-side web page composition," in *Proceedings of the 7th IEEE International Conference on Software Engineering and Service Science*, Beijing, pp. 45-53, 2016.

[18] PJAX library [Internet]. Available: https://github.com /defunkt/jquery-pjax.

[19] M. Pilgrim. Dive into HTML5 [Internet]. Available: http://diveintohtml5.info/.

[20] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel, "Performance comparison of middleware architectures for generating dynamic web content," in *Proceedings of the 4th ACM/IFIP/USENIX Middleware*, Rio de Janeiro, pp. 242-261, 2003.



오재원(Jaewon Oh)

2004년 서울대학교 계산통계학과(학사, 석사, 박사)
2004년 ~ 2007년 삼성전자 책임연구원
2007년 ~ 현재 가톨릭대학교 컴퓨터정보공학부 부교수
※관심분야 : Software Engineering, Web Engineering, System Software

안우현(Woo Hyun Ahn)

1996년 경북대학교 전자공학과 (학사)
1998년 KAIST 전기 및 전자공학과 (석사)
2003년 KAIST 전자전산학과 (박사)
2003년 ~ 2005년 삼성전자 책임연구원
2006년 ~ 현재 광운대학교 소프트웨어학부 교수
※관심분야 : 운영체제, 임베디드시스템, 시스템 보안

김태공(Taegong Kim)

1994년 서울대학교 계산통계학과(학사, 석사, 박사)
2003년 ~ 2004년 The University of Texas at Dallas 방문교수
1990년 ~ 현 재 인제대학교 컴퓨터공학부 부교수
※관심분야 : Software Engineering, Refactoring, Code Smell Detection