JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 22, No. 3: 458~470, Mar. 2018

자바 웹 앱에서 웹 컴포넌트와 웹 자원의 의존 관계를 자동으로 추출하는 기법

오재원[1] · 이승현[1] · 김아형[1] · 안우현[2*]

An Automatic Extraction Scheme of Dependency Relations between Web Components and Web Resources in Java Web Applications

Jaewon Oh[1] · Seunghyun Lee[1] · Ah Hyoung Kim[1] · Woo Hyun Ahn[2*]

[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

요 약

웹 앱의 요구사항이 복잡해지고 빠르게 변화하고 있어, 웹 앱의 유지보수가 더욱더 중요해지고 있다. 그렇지만, 웹 앱은 문서화가 충분하게 이루어지지 않아 유지보수가 어렵다. 그래서 효과적인 웹 앱의 유지보수를 위해 웹 페이지 생성 시 일어나는 내부 행위를 추상화한 모델을 추출할 필요가 있다. 기존 연구는 상호 작용하는 웹 컴포넌트(JSP, 서블릿 등)를 식별하지만, 웹 자원(이미지, CSS 파일, 자바스크립트 파일 등)을 식별하지 못하거나, 웹 컴포넌트와 웹 자원 사이의 의존 관계를 추출하지 못한다. 본 논문은 자바 웹 앱을 동적 분석하여 이러한 의존 관계를 추출하고, 그래프 모델로 표현하는 방법을 제안한다. 그리고 오픈 소스 웹 앱을 대상으로 실험하여 제안하는 기법의 활용 가능성을 검증한다.

ABSTRACT

As the requirements of web apps become complex and rapidly changing, the maintenance of web apps becomes more important. However, web apps have a problem that more often than not there is not enough documentation to understand and maintain them. Thus, their effective maintenance requires models that represent their internal behavior occurring when they dynamically generate web pages. Previous works identify web components (such as JSPs and Servlets) as participants in the behavior but not web resources (such as images, CSS files, and JavaScript files). Moreover, they do not identify dependency relations between web components and web resources. This paper dynamically analyzes Java web apps to extract such dependency relations, which are included in our graph model for page generation. Case studies using open-source web apps show the applicability of the proposed approach.

키워드 : 웹 컴포넌트 상호 작용, 웹 컴포넌트, 웹 자원, 의존 관계, 웹 공학

Key word : Web Component Collaboration, Web Component, Web Resource, Dependency Relation, Web Engineering

Received 27 December 2017, **Revised** 5 January 2018, **Accepted** 16 January 2018

*** 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 http://doi.org/10.6109/jkiice.2018.22.3.458 pISSN:2234-4772

Ⅰ. 서 론

웹 앱의 요구사항이 나날이 복잡해지고[1, 2] 빠르게 변화하고[3] 있다. 또한 웹 앱의 빠른 출시에 대한 요구로[4, 5] 인해 웹 앱에 대한 문서화가 충분히 이루어지지 않은 경우가 적지 않다[2, 4]. 아울러 웹 개발자의 이직이 잦다[4]. 이러한 이유 등으로 웹 앱 개발자는 유지보수에 어려움을 겪고 있다[1, 5, 6]. 그렇지만 웹 앱의 이해를 위한 연구가 많지 않다[6].

웹 앱의 효과적인 유지보수를 위해서는 웹 앱의 페이지 생성 과정을 추상화한 모델을 추출할 필요가 있다[7, 8]. 페이지 요청이 발생하면, 웹 서버에서 여러 웹 컴포넌트(JSP, 서블릿 등)가 상호 작용하여 결과 페이지가 생성되고 웹브라우저로 반환된다. 이 페이지가 웹브라우저에서 로드되는 과정에서 새롭게 웹 자원에 대한 요청이 발생할 수 있다. 예를 들면, 페이지가 이미지, 자바스크립트 파일, CSS 파일을 필요로 하는 경우, 이러한 웹 자원 각각을 위해 HTTP 요청이 이루어진다. 따라서 페이지 생성 과정을 표현하는 모델은 위에서 설명한 웹 컴포넌트, 웹 자원과 이들 사이의 관계를 표현할 필요가 있다.

페이지 생성 과정을 표현하기 위한 모델을 정의하고 추출하기 위해 기존에 연구 [7-9]가 제안되었다. 이들 연구는 동적 분석[7] 혹은 정적 분석[8, 9]을 활용하여 웹 컴포넌트와 이들 사이의 상호 작용을 추출한다. 즉, 자바 웹 앱을 구성하는 웹 컴포넌트인 서블릿, JSP와 이들 간의 상호 작용을 복구한다. 하지만 이 연구들은 웹 자원(이미지, CSS 파일, 자바스크립트 파일 등)을 식별하지 못하고 아울러 웹 자원과 웹 컴포넌트 사이의 상호 작용도 추출하지 못한다.

기존 연구와 달리 페이지 생성 과정을 위한 모델이 웹 자원과 웹 컴포넌트 간의 상호 작용까지 포함할 경우의 이점은 다음과 같다. 첫째, 웹 자원과 의존 관계를 맺는 개체가 보다 더 작은 수준에서 파악될 수 있다. 현재까지 연구는 웹 자원을 식별하지 않거나 식별하더라도 웹 자원을 필요로 하는 기본 개체를 페이지로 간주하지만, 앞으로는 웹 컴포넌트 단위로 의존 관계를 파악할 수 있어 더 세밀하게 웹 앱을 관리할 수 있다. 둘째, 요구사항 변경 시 특히 웹 자원을 변경할 때, 파급 효과를 페이지 단위가 아니라 웹 컴포넌트 단위로 좀 더 세밀하게 분석할 수 있다. 셋째, 웹 자원의 중복을 웹 컴포넌트 단위에서 분석하고 관리할 수 있다. 넷째, 웹 앱 아키텍처 적합성 검사[10] 시 아키텍처 불일치 문제[10, 11]를 웹 컴포넌트와 웹 자원 관계 측면에서 수행할 수 있어 검사가 더욱 효과적일 수 있다.

본 논문은 웹 컴포넌트, 웹 자원과 이들 간의 상호 작용을 표현하는 페이지 생성 그래프 모델을 제안한다. 또한 자바 웹 앱으로부터 이 모델을 자동적으로 추출하기 위한 동적 분석 방법을 제안한다. 제안하는 모델과 추출 방법의 유효성을 검증하기 위해 오픈 소스를 대상으로 실험한다. 본 논문은 배경, 페이지 생성 그래프 모델, 모델 추출 방법, 추출 방법 적용 사례, 결론 순서로 서술된다.

Ⅱ. 배 경

제안하는 기법의 이해를 위해 웹 컴포넌트 간의 상호 작용을 표현하는 기존 연구 [7-9]에 대해 설명한다. 특히, [7]에서 제안하는, 요청 처리 경로 모델(collaboration model)에 대해 자세히 설명한다.

2.1. 요청 처리 경로 모델

사용자는 새 페이지를 요청하기 위해 폼을 서브밋하거나 하이퍼링크를 클릭한다. 이러한 요청이 발생하면 웹 서버에서는 여러 웹 컴포넌트(JSP, 서블릿, HTML)가 상호 작용하여 새 페이지를 생성하고 웹브라우저로 반환한다. 이러한 상호 작용을 표현하기 위해 연구 [7]에서 요청 처리 경로 모델을 제안하였다. 페이지 요청을 처리하기 위한 모델은 방향 그래프로 표현된다. 그림 1(B)에 요청 처리 경로 모델의 예가 나와 있다.

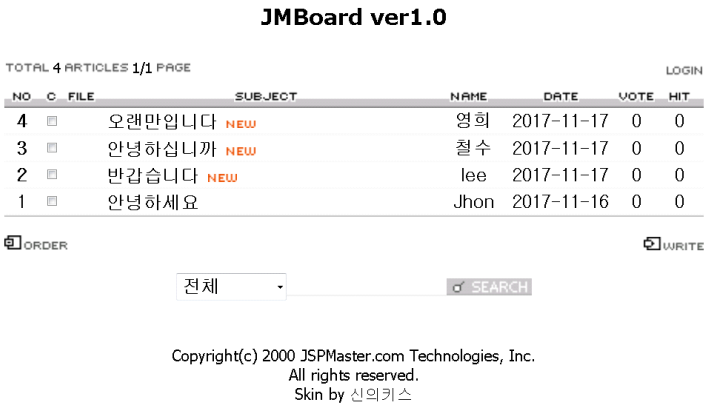
그림 1은 JMBoard(http://happycgi.com/8168)에서 게시물 목록을 읽기 위해 하이퍼링크를 클릭할 때 일어나는 웹 컴포넌트간의 상호 작용(그림 1(B))과 생성된 결과 페이지(그림 1(A))를 보여준다. JMBoard는 게시판을 생성하고, 관리하고, 삭제하는 기능을 제공하는 오픈 소스 웹 앱이다.

그림 1에서 표현하는 상호 작용을 기술하면 다음과 같다. 사용자가 게시물 목록을 읽는 요청을 하면, 이 요청이 서버의 웹 컨테이너에게 전달된다. 웹 컨테이너는 이 요청을 처리하는 컨트롤러[12]인 JMBoard 서블릿을 호출한다. JMBoard가 게시물 목록 획득과 관련한 비즈

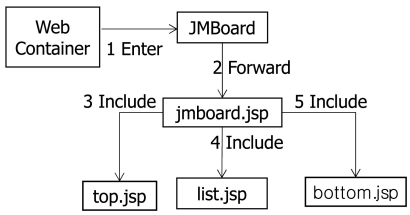
니스 로직을 수행한다. 그러고 나서 JMBoard가 결과 출력을 위해 뷰[12]인 jmboard.jsp를 Forward로 호출한다. 이 JSP는 top.jsp, list.jsp, bottom.jsp 뷰를 차례로 포함하여 결과 페이지를 생성한다. 이 결과 페이지는 웹 컨테이너를 통해 웹브라우저에 전달된다.

그런데 위 페이지가 웹브라우저에 로드되는 과정에서 새로운 웹 자원이 필요하여 이들 각각을 위한 HTTP 요청이 발생한다. 그림 1(C)는 그림 1(A) 페이지 생성을 위해 필요한 웹 자원이 목록이다. 기존 연구 [7-9]에서는 서버 측면에서 발생하는 상호 작용만을 고려하기 때문에 그림 1(C)의 웹 자원 요청이 모델에 표현되지 않는다.

효과적인 유지보수를 위해서는 사용자의 페이지 요청을 만족시키기 위해 필요한 웹 자원을 파악하고 문서화할 필요가 있다. 그래서 본 논문은 그림 1(B)와 그림 1(C)를 결합한 그림 2를 제안한다. 그림 2를 통해 웹 페이지 단위가 아니라 웹 컴포넌트 단위로 각 웹 컴포넌트가 어떤 웹 자원을 필요로 하는지 확인할 수 있다. 이 예제에서는 top.jsp 뷰가 style.css를 사용하며, style.css는 head_bg.gif를 사용한다. 나머지 웹 자원은 list.jsp에서 사용된다.



(A) Page



(B) Collaboration graph

No.	Web resource	No.	Web resource
1	head_bg.gif	11	head_subject.gif
2	style.css	12	head_file.gif
3	setup_articles.gif	13	i_admin.gif
4	setup_pages_nowpage.gif	14	old_head.gif
5	setup_total.gif	15	jpeg.gif
6	head_c.gif	16	head_vote.gif
7	head_no.gif	17	head_hit.gif
8	member_login.gif	18	i_write.gif
9	head_date.gif	19	search.gif
10	head_name.gif		

(C) Web resources

Fig. 1 Page, web component collaboration, and web resources of JMBoard for article list reading

2.2. 기존 웹 컴포넌트 상호 작용 모델

기존 연구 [8, 9]도 [7]과 유사하게 페이지 요청 시 웹 서버에서 발생하는 웹 컴포넌트 간의 상호 작용을 표현하기 위한 모델을 제시하고 모델을 추출하는 방법을 제안하였다. [7]은 동적 분석을 통해 모델을 추출하며 [8-9]는 정적 분석을 통해 모델을 추출한다. 그렇지만 [8-9] 또한 [7]과 비슷하게 웹 자원을 고려하지 않는다.

정적 분석은 소스 코드를 파싱하여 웹 앱을 구성 요소로 나누고 이해하려는 시도이다. 반면에 동적 분석은 웹 앱을 실제로 실행하여 그 행동을 분석하여 웹 앱을 이해하려는 접근법이다. [5, 7]에서 지적하듯이 여러 프로그래밍 언어(HTML, 자바스크립트, CSS, XML, Java, JSP 등)를 함께 사용하는 웹 앱의 이해를 위해 정적 분석만을 활용하는 것은 한계가 있다. 즉 정적 분석만으로는 동적으로 생성되는 웹 앱 코드와 여러 언어의 상호 작용을 분석하기 어렵다. 또한 정적 분석에서는 여러 언어를 파싱할 필요가 있는데 이 작업 또한 비용이 많이 든다.

본 논문에서는 동적 분석을 통해 그림 2와 같은 그래프 모델을 생성하는 것을 목표로 한다. 즉, 각 웹 자원을 필요로 하는 웹 컴포넌트와 웹 자원 사이에 의존 관계를 생성하는 것이다.

Ⅲ. 페이지 생성 그래프 동적 추출

본 논문은 [7]에 기반을 두고 동적 분석을 통해 그림 2와 같은 그래프를 추출하는 것을 목표로 한다.

3.1. 페이지 생성 그래프

페이지 한 개를 생성하기 위해 필요한 웹 컴포넌트, 웹 자원과 이들 사이의 관계를 표현하기 위해 아래 (정의 1)에서 그래프 모델을 제안한다. 이 그래프 모델은 기존 연구 [7]에서 제안한 요청 처리 경로 모델을 확장한다. 그림 2가 이 그래프 모델의 한 인스턴스이다.

(정의 1). (페이지 생성 그래프, Page Generation Graph with Resource Added, PG2RA, 그림 2 참고) 페이지 요청 r이 주어질 때, r의 페이지 생성 그래프 $Gr(V, E, t, o)$는 방향 그래프이며 다음 조건을 만족한다.

• V: r의 처리 과정에서 실행되는 웹 컴포넌트(JSP, 서블릿, HTML등)와 웹 자원(이미지, 자바스크립트 파일, CSS 파일, 폰트 파일 등)의 집합이다.

• E: V에 속하는 두 노드 사이의 관계를 표현한다. V에 속하는 v_i, v_j에 대해, v_i가 v_j를 호출하면 $<v_i, v_j>$가 E에 속한다. 단, 다섯 가지 호출 타입(*Enter, Forward, Include, Redirect, Use*)이 존재하며, 아래에서 구체적으로 설명한다.

• $t: E \rightarrow \{Enter, Forward, Include, Redirect, Use\}$. t는 E에 속하는 각 간선의 호출 타입을 구하는 함수이다. 예를 들면, E에 속하는 $ei(<v_j, v_k>)$에 대해, v_j가 v_k를 include 액션으로 호출하면 $t(ei)=Include$가 된다. 가능한 호출 타입은 다음과 같다.

- *Enter*: 페이지의 생성 요청을 처리할 때 웹 컨테이너에서 첫 웹 컴포넌트로 가는 간선 타입이다.

- *Forward*: forward 액션을 호출하는 웹 컴포넌트에서 호출되는 웹 컴포넌트로 가는 간선 타입이다.

- *Include*: include 액션을 호출하는 웹 컴포넌트에서 호출되는 웹 컴포넌트로 가는 간선 타입이다.

- *Redirect*: redirect 액션을 호출하는 웹 컴포넌트에서 목적지 URL을 나타내는 가상의 웹 컴포넌트로 가는 간선 타입이다. 가상 컴포넌트를 두는 이유는 redirect를 받는 웹 컴포넌트가 웹 서버에 실제로 존재하는 것이 아니지만, 다른 호출 타입의 정의와 일관성을 맞추기 위해서이다.

- *Use*: 웹 자원을 필요로 하는 웹 컴포넌트 혹은 웹 자원에서 해당 웹 자원으로 가는 간선 타입니다.

• $o: E \rightarrow \{1, 2, ..., |E|\} \cup \{undefined\}$. o는 E에 속하는 각 간선의 호출 순서를 반환하는 함수이다. 페이지 생성 그래프를 구성하는 웹 컴포넌트의 실행 순서 정보를 표현한다. 그렇지만 웹 컴포넌트와 웹 자원, 웹 자원들 간의 간선에 대해서는 함수 값을 정의하지 않는다. 즉, Use 관계에 대해서는 실행 순서를 정의하지 않는다. 이유는 페이지 생성 시 서버 측에서 웹 컴포넌트가 실행되는 순서는 페이지의 올바른 생성을 위해 중요한 정보이지만, 웹 컴포넌트가 필요로 하는 웹 자원은 페이지 로드 완료 시까지 로드될 필요가 있지만, 웹 자원간의 로드 순서는 중요하지 않기 때문이다.

예를 들면, E의 원소 중 가장 먼저 실행되는 호출을 e_i라 하면, $o(e_i)=1$이다. E의 원소 중 두 번째로 실행되는 호출을 e_j라 하면 $o(e_j)=2$이다. 그리고 $t(e_k)=Use$이면 $o(e_k)=undefined$이다.

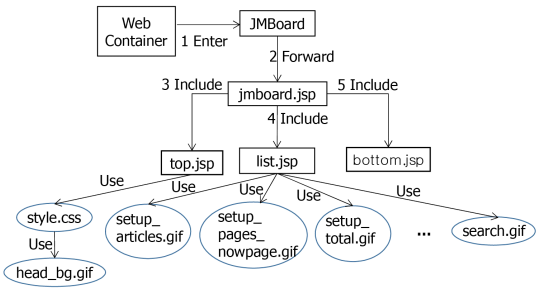


Fig. 2 Web component and web resource collaboration of JMBoard for article list reading

3.2. 페이지 생성 그래프 추출

3.2.1. 동적 추출 방법 개요

본 논문에서는 앞에서 정의한 페이지 생성 그래프(PG2RA)를 자바 웹 앱으로부터 추출하기 위해서 동적 분석 방법을 사용한다. 페이지 생성 시 웹 서버에서 일어나는 웹 컴포넌트 사이의 상호 작용을 표현하고 추출하기 위해서 기존 연구 [7]에서 제안한 방법을 사용한다. 본 논문은 [7]의 요청 처리 경로 모델에 웹 자원을 노드로 추가하고, 웹 컴포넌트와 웹 자원 사이의 Use 관계를 간선으로 추가하는 방법을 제안한다. 이 방법은 기존 연구 [7]과 같이 동적 분석을 이용한다.

웹 컴포넌트와 웹 컴포넌트 간의 관계는 웹 서버에서 식별된다[7]. 이와 달리 본 연구에서 고려하는, 웹 자원과 Use 간선의 식별과 추가는 웹브라우저에서 이루어진다. 이를 위해서 사용자가 요청 시 웹 서버에서 결과 페이지가 생성될 때, 이 페이지에 Use 관계를 식별하기 위한 코드 C_{use}를 추가한다. 이 후 웹브라우저에서 결과 페이지를 로드할 때 이 페이지가 추가적으로 필요로 하는 웹 자원을 식별하고, C_{use}를 참조하여 이 웹 자원을 구체적으로 필요로 하는 웹 컴포넌트를 파악하여 PG2RA를 완성한다.

3.2.2. 아키텍처

그림 3에 PG2RA를 추출하기 위한 시스템 아키텍처가 나와 있다. 아키텍처를 구성하는 주요 요소를 페이지 요청을 처리하는 순서로 설명하면 다음과 같다.

사용자가 새로운 페이지를 요청한다(그림 3의 단계 1). 기존 연구 [7]를 따라 웹 컴포넌트와 이들 사이의 상호 작용을 식별한다(단계 2). 서블릿 필터와 래퍼 [13]를 활용하여 결과 페이지 안에 웹 컴포넌트의 실행 정보(웹 컴포넌트가 실행을 시작한 시점과 종료한 시점)를 추가한다(단계 3). 추가된 정보는 Use 관계 식별을 위한 코드이며, 앞에서 C_{use}라고 불렀다. 웹 컴포넌트의 상호 작용이 끝나고, 결과 페이지가 웹브라우저에 도착하고 로드되기 시작한다(단계 4, 5). 결과 페이지가 추가적으로 필요로 하는 웹 자원을 요청한다(단계 6). 웹브라우저에서 웹 자원 식별 모듈(Resource extractor)이 이 웹 자원을 식별한다(단계 7). 페이지 로드가 완료된 후 Use 관계 식별 모듈(Resource dependency extractor)이 페이지 DOM(Document Object Model) 트리를 접근하여 웹 자원과 웹 컴포넌트 사이의 Use 관계를 식별하고 PG2RA에 추가한다(단계 8).

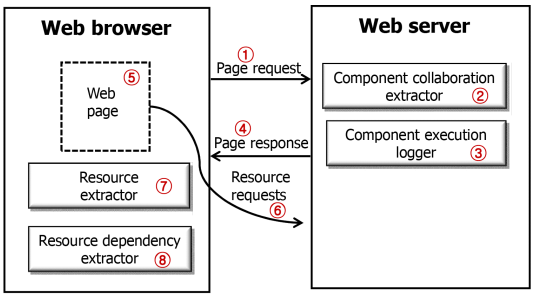


Fig. 3 Architecture for extraction of dependency relations between web components and web resources

3.2.3. 웹 컴포넌트의 실행 블록

Use 관계를 식별하기 위해 웹 서버에서 결과 페이지를 생성하는 과정에서 원본 결과 페이지에 웹 컴포넌트 실행 정보(C_{use})를 추가한다. 이를 위해 우선 웹 컴포넌트 i의 실행 블록(execution block)을 다음과 같이 정의한다.

(정의 2). (웹 컴포넌트의 실행 블록) 웹 컴포넌트 i가 주어질 때, i의 실행 블록은 결과 페이지 중 i가 생성한 코드와 이 코드를 결과 페이지 내의 다른 코드와 구별하기 위한, 경계를 명시하는 코드로 구성된다.

예를 들면 그림 2는 PG2RA를 보여준다. 그림 1(A)는 그림 2의 과정을 통해 만들어진 결과 페이지를 보여준다. 그렇지만 그림 1(A)만을 이용해서는 어떤 웹 컴포넌트가 결과 페이지에서 어떤 부분의 코드를 생성했는지 식별할 수 없다.

이와 달리 그림 4는 결과 페이지가 웹 컴포넌트 실행 블록을 지원할 경우를 보여준다. 그림 4(B)에서 점

선으로 둘러싼 네모가 웹 컴포넌트의 실행 블록을 표현한다. 이를 통해 페이지가 Include 관계를 기준으로 계층적으로 구성된 모습을 확인할 수 있다. 예를 들면 웹 컴포넌트 C2는 웹 컴포넌트 C에 포함되며, 웹 컴포넌트 C21을 포함하며 웹 자원 R21을 사용한다.

그림 4(B)를 보면 웹 컴포넌트 S가 출력하는 코드가 없다. 그 이유는 다음과 같다. 일반적으로 자바 웹 앱에서는 사용자 요청에 대응하는 비즈니스 로직을 수행한 후 결과 페이지 작성을 위해 출력을 담당하는 웹 컴포넌트로 Forward한다(그림 4(A) 단계 2 참조). Forward 이전에 결과 페이지를 출력하려는 시도가 있으면 익셉션(exception)이 발생한다.

그리고 자바 웹 앱에서는 결과 페이지 출력을 모듈화하기 위해 Include 호출을 사용한다. 그림 4(A)의 경우 웹 컴포넌트 C의 관리 하에서 출력 작업이 C1, C2, C21 웹 컴포넌트로 모듈화되어 있음을 알 수 있다. 여기서 편의상 모든 출력 컴포넌트를 포함하여 관리하는 웹 컴포넌트 C를 이 페이지 요청의 메인 웹 컴포넌트라고 하자. 그리고 메인 웹 컴포넌트가 생성하는 실행 블록을 메인 실행 블록이라고 하자.

3.2.4. 추출 알고리즘: 실행 블록 정보가 추가된 결과 페이지 생성

웹 자원과 Use 관계를 추출하는 과정은 크게 두 가지(결과 페이지 생성과 PG2RA 완성)로 나누어 볼 수 있다. 이번 절에서는 웹 컴포넌트 실행 블록 정보가 추가된 결과 페이지를 생성하는 방법에 대해 설명한다. 다음 절에서 PG2RA를 완성하는 방법에 대해 설명한다.

그림 4(B)와 같은, 웹 컴포넌트 실행 블록을 표현하는 결과 페이지 생성을 위한 알고리즘이 그림 5에 나와 있다. 이 알고리즘은 그림 3의 웹 컴포넌트 실행 로거(Component execution logger)가 사용한다. 기본적으로 자바 웹 컴포넌트의 필터 기술을 사용하여, 웹 컴포넌트 실행 블록을 결과 페이지에 표시한다. 일반적으로 자바 웹 컴포넌트인 JSP, 서블릿은 사전 필터(pre-filter)와 사후 필터(post-filter)를 가질 수 있다. 각 필터는 웹 컴포넌트가 실행하기 전과 후에 실행되는 모듈이며, 웹 컴포넌트 개발자가 원하는 작업(예를 들면 암호화, 인코딩 등)을 수행할 수 있다.

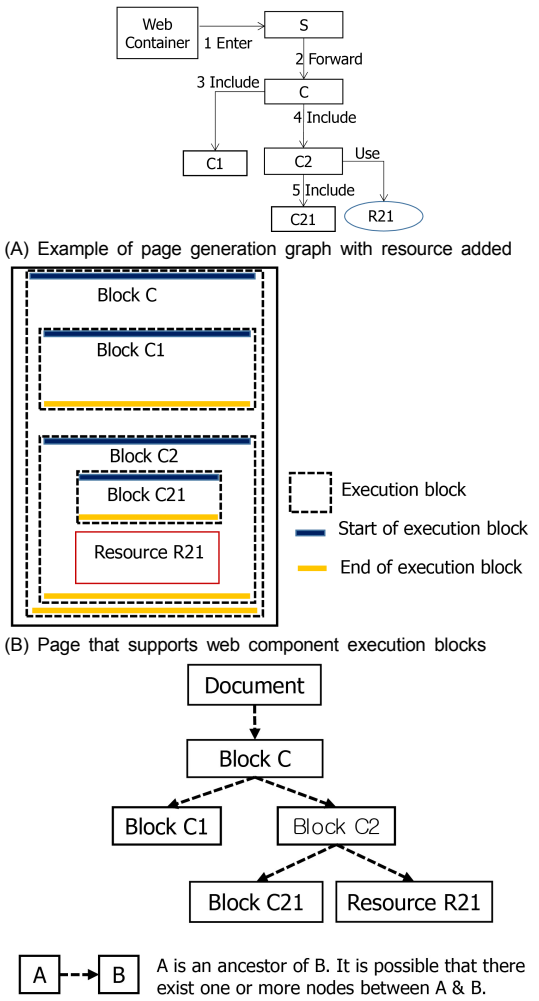


Fig. 4 Web component execution blocks for extraction of dependency relations between web components and web resources

그림 5 알고리즘에서는 웹 컴포넌트 *i*가 실행하기 직전에 실행되는 사전 필터에서 *i*가 수행을 시작한다는 사실을 결과 페이지에 추가한다(그림 5 단계 7, 8). 그리고 *i*가 실행을 마친 후 실행되는 사후 필터에서 *i*가 수행을 마친다는 사실을 페이지에 추가한다(그림 5 단계 9, 10). 이 두 시점 사이에는 *i*가 원래 출력하는 코드가 그대로 결과 페이지에 삽입된다.

그런데 메인 웹 컴포넌트가 수행되기 전에 Include, Forward 호출이 가능하다. 이러한 호출은 출력 작업이

아니라 비즈니스 로직의 처리를 위한 것이다. 즉, 결과 페이지에 영향을 미치지 않는다. 따라서 메인 웹 컴포넌트 시작 시점을 식별하고, 그 이후에 이루어지는 웹 컴포넌트 상호 작용만을 고려해야 한다.

Add component execution blocks to a resulting page for a request *r*

1: let *curResultingPage* be a page currently being created for request *r* and initially empty
2: let *BLOCK_START* be a start tag for an HTML element representing an execution block //e.g. set *BLOCK_START* to " "
3: let *BLOCK_END* be an end tag for an HTML element representing an execution block //e.g. set *BLOCK_END* to " "
4: **if** (r is routed to a pre-filter of Forward call)
5: set *curResultingPage* to be empty
6: **end if**
7: **if** (*r* is routed to a pre-filter of Include, Forward, or Enter call)
8: append to *curResultingPage, BLOCK_START* tag with block *id*
9: **else if** (*r* is routed to a post-filter of Include or Enter call)
10: append *BLOCK_END* to *curResultingPage*
11: **else if** (*r* is routed to a post-filter of the last Forward call)
12: append *BLOCK_END* to *curResultingPage*
13: **end if**

Fig. 5 Algorithm for generating page with execution blocks using Servlet filter

메인 컴포넌트 시작 시점을 Forward 호출 관점에서 파악할 수 있다. 페이지 요청에 대해 *n* 개의 Forward 호출이 가능하다(n≥0). 이 경우 마지막 *n* 번째 Forward 호출 이후 페이지 출력 작업이 실제로 시작된다.

*n=0*의 경우는 Forward 호출이 전혀 없는 경우이다. 이 경우는 Enter 호출의 사전과 사후 필터에서 메인 실행 블록을 결과 페이지에 표시한다(단계 7-10). 이 두 시점 사이에 이루어지는 Include 호출은 이 메인 실행 블록 안에 놓이게 된다.

n≥1의 경우는, n 번째 Forward 호출이 이루어지면 사전 필터에서 우선 이 호출 이전에 수행된 사전, 사후 필터 작업의 결과를 리셋한다(단계 4-6). 그리고 n 번째 호출의 사전 필터(단계 7, 8)와 사후 필터(단계 11, 12)에서 메인 실행 블록을 결과 페이지에 표시한다. 이 두 시점 사이에 이루어지는 Include 호출은 이 메인 실행 블록 안에 놓이게 된다.

그림 5 알고리즘을 이용하고 그림 4(A)와 같은 동작을 하는 페이지 요청이 있으면, 그림 4(B)와 같은 결과 페이지가 생성된다. 실행 블록 C는 Forward 호출의 사전, 사후 필터에서 생성되며, 나머지 실행 블록은 Include 호출의 사전, 사후 필터를 통해 생성된다.

웹 컴포넌트 *i*의 시작과 끝을 표시하기 위해서는 구현 수준에서 여러 대안이 있을 수 있다. 본 연구에서는 시작 시에는 * *을, 마칠 때에는 * *를 사용했다. *id* 속성은 각 웹 컴포넌트를 구별하기 위한 식별자이다(그림 5 단계 8).

그림 4(C)는 그림 5 알고리즘으로 생성한 결과 페이지(그림 4(B))가 웹브라우저로 로드된 후 만들어진 DOM 트리를 보여준다. 여기서 웹 컴포넌트 실행 블록 *i*(그림 4(C)의 C2)에서 필요로 하는 웹 자원을 요청하는 태그(그림 4(C)의 R21)는 실행 블록 *i*의 경계를 표현하는 태그의 후손임을 알 수 있다. 다음 절에서 이 사실을 이용하여 웹 컴포넌트와 웹 자원 사이의 Use 관계를 식별한다.

3.2.5. 추출 알고리즘: 웹 자원, 의존 관계 식별을 통한 PG2RA 완성

이번 절에서는 웹브라우저에서 웹 자원과 Use 관계를 추가하여 PG2RA를 완성하는 방법에 대해 설명한다. 이 과정에서 우선 웹 자원을 파악하기 위해 웹 자원 식별 모듈(그림 3의 Resource extractor)을 사용한다. 이 모듈을 구현하기 위해 파이어폭스의 WebExtension[14]에서 제공하는 백그라운드 태스크 기술을 활용한다. 이 기술을 활용하여 백그라운드로 페이지 로딩 과정을 모니터링하여 이 과정에서 발생하는 HTTP 요청을 식별하고 수집한다.

페이지 로드가 완료된 후, Use 관계 식별 모듈(그림 3의 Resource dependency extractor)이 수행된다. 이 모듈은 웹 자원 식별 모듈로부터 웹 자원을 요청하는 HTTP 요청 리스트를 입력받는다. 이 모듈은 다음과 같은 일을 수행한다. 각 HTTP 요청을 발생시킨 태그 *t*를 식별한다. 그리고 태그 *t*를 포함하는 웹 컴포넌트 *c*의 실행 블록을 식별한다. 웹 컴포넌트 *c*와 태그 *t*를 연결하는 Use 관계를 PG2RA에 추가한다.

*t*를 포함하는 블록이 두 개 이상일 수 있다. 예를 들면 그림 4(C)에서 Resource R21을 포함하는 실행 블록에는 C2와 C가 있다. 그렇지만 정확하게 R21을 사용하는 블록은 가장 안쪽 블록인 C2이다. 그래서 C2와 R21 사이의 Use 간선을 PG2RA에 추가한다.

정리하면 PG2RA에서 웹 자원 *r*과 Use 관계를 생성하는 웹 컴포넌트 *c*는 다음 조건을 만족한다. *r*을 요청하는 태그를 t_r라고 하자. 웹 컴포넌트 *c*의 실행 블록을 나타내는 태그를 t_c라 하자. t_c는 DOM 트리에서 t_r의 조상이며, t_r로부터 가장 가까운, 실행 블록을 나타내는 노드이다.

위 과정을 위해 Use 관계 식별 모듈은 DOM 트리를 검색할 필요가 있다. 이를 위해 파이어폭스의 WebExtension에서 제공하는 콘텐트 스크립트 기술을 활용한다. 콘텐트 스크립트는 웹 서버로부터 넘겨받아 현재 로딩된 페이지를 접근할 수 있다.

그림 6에 Use 관계를 식별하는 알고리즘이 나와 있다. 트리의 순회(traverse) 방법 중 후위 순회(postorder traverse)를 전체적으로 한 번 수행한다. 웹 자원을 요청하는 HTTP 요청별로 DOM 트리를 매번 검색하는 것보다 효율적이기 때문에 이 순회 방법을 선택하였다.

현재 순회 진행 중인 실행 블록 *curExeBlock*(그림 6의 단계 1)을 식별하기 위해, 실행 블록 스택 *ExeBlockStack*(단계 2)를 사용한다. 순회 도중 웹 컴포넌트 실행 블록 *i*를 처음 만나면(단계 4), *ExeBlockStack*에 *i*를 푸시하여 현재 탐색 중인 실행 블록을 i로 변경한다(단계 5, 6). 푸시하는 이유는 이 후 순회를 계속하면 *i*를 방문(단계 11)하기 직전(단계 8-10)까지 먼저 방문하는 다른 노드는 *i*를 루트로 하는 서브트리에 속하며, 이 서브트리는 블록 *i*의 웹 컴포넌트가 출력하는 코드이거나 이 블록에 포함되는 다른 실행 블록이기 때문이다.

웹 자원을 요청하는 노드 *r*를 방문하면(단계 14) 현재 실행 블록이 *r* 자원을 사용하는 것이기 때문에 현재 실행 블록과 *r* 사이에 Use 관계를 생성한다(단계 15).

실행 블록 *i*를 방문하면(단계 16), 실행 블록 *i* 내부를 모두 순회한 것이기 때문에 *ExeBlockStack*에서 현재 실행 블록을 팝한다(단계 17). 이 후 스택의 최상위에 있는 원소 *b*는 *i*를 포함하는 실행 블록이며, 새로운 현재 실행 블록이 된다(단계 18-19). 예를 들면 그림 4에서 실행 블록 C2를 방문한다면, C21과 R21 등 C2 내부 노드를 모두 이미 방문한 것을 의미한다. 따라서 이제 현재 실행 블록을 C2를 포함하는 조상 노드로 변경해야 하며, 이 노드는 *ExeBlockStack*의 top 위치에 존재한다.

Find Use relations of page generation graph G_r for a request *r*

```
1: let curExeBlock be an execution block currently being traversed for request r
2: let ExeBlockStack be a stack of execution blocks to identify curExeBlock, which is a top element of ExeBlockStack
3: procedure postorder_traverse(node n)
4:   if (n represents an execution block i ) then
5:     push i on ExeBlockStack
6:     set curExeBlock to i
7:   end if
8:   for each child c of n 's children
9:     postorder_traverse(c)
10:  end for
11:  visit(n)
12: end procedure
13: procedure visit(node n)
14:  if (n represents a request of web resource r ) then
15:    add <curExeBlock, r > as an Use relation to G_r
16:  else if (n represents an execution block i )
17:    pop an element from ExeBlockStack
18:    set curExeBlock to a top element of ExeBlockStack
19:  end if
20: end procedure
```

Fig. 6 Algorithm for finding Use relations, based on postorder traverse of DOM trees

현재 웹 자원을 요청하는 HTML 태그로 고려하는 것은 <*img*>, <*script*>, <*link*>이다. 이들의 *src, href* 속성에 웹 자원을 요청하는 URL이 표현된다.

웹 자원 식별 모듈에서 수집한 HTTP 요청 리스트가 후위 순회 시 발견한 HTTP 요청 리스트와 다를 수 있다. 예를 들면 자바스크립트에서 AJAX를 이용하여 HTTP 요청을 하게 되면, 웹 자원 식별 모듈에서는 식별되지만 DOM 트리 탐색 시에는 발견되지 않을 수 있다. 왜냐하면 DOM 트리의 노드로 AJAX 요청이 표현되지 않을 수 있기 때문이다. 이런 경우는 요청을 발생한 실행 블록을 정확하게 식별할 수 없어, 이 요청을

메인 실행 블록과 해당 웹 자원 사이의 Use 관계로 표현한다. 예외적으로 HTTP 요청의 *referer* 속성으로 Use 관계를 식별할 수 있는 경우가 있다. 예를 들면 CSS 파일 *f*에서 이미지 *m*을 사용하는 경우, *m* 요청 시 HTTP 요청의 *referer* 헤더 값으로부터 *f*가 *m*을 요청했다는 것을 알 수 있다. 본 논문에서 이런 경우 <*f,m*>을 PG2RA에 Use 관계로 추가한다. 그림 2에서 간선 <style.css, head_bg.gif>가 이런 예에 속한다.

Ⅳ. 실험 및 결과 분석

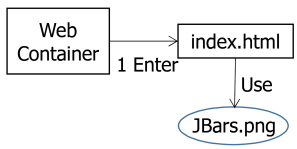
유효성 검증을 위해 세 개의 오픈 소스 웹 앱(JBars, JMBoard, JPetStore)을 대상으로 실험하였다. 이 웹 앱을 선택한 이유는, 첫째 웹 공학 분야의 기존 연구 [7, 15-19]에서 유효성 검증을 위해 사용된 앱이기 때문이다. 둘째, 유명 오픈 소스 저장소에 코드가 공개되어 있다. 셋째, HTML, JavaScript, 서블릿, JSP, 표준 액션 태그, AJAX 등으로 구성되어 있어 Java 웹 앱의 전형적인 구성을 따르기 때문이다.

첫 번째 웹 앱은 바코드를 생성하는 JBars (https://sourceforge.net/projects/jbars/?source=directory)이다. 바코드 생성을 위해 사용자가 자료를 입력하는 페이지와 이 페이지의 PG2RA가 그림 7에 나와 있다. 그림 7(A) 페이지에서 데이터를 입력하고 생성 버튼(Barcode Generator)을 누르면 바코드가 생성된다. 그림 7(B)의 PG2RA에서 알 수 있듯이, index.html 웹 컴포넌트가 입력 UI를 출력하는 역할을 담당한다. 그리고 이 컴포넌트는 JBars.png 이미지 파일을 사용하여 JBars 웹 앱의 로고를 페이지 상단에 표시한다.

둘째, 2.1절에서도 소개된 JMBoard 웹 앱은 게시판을 생성하고 관리할 수 있게 한다. 또한 생성한 게시판에 게시글을 올릴 수 있게 한다.

JMBoard에 관한 그림 8은 그림 1(A)에서 첫 번째 게시글을 선택하여 게시글 읽기를 요청한 결과이다. 그림 8(A)의 결과 페이지 생성을 위해 관련 웹 컴포넌트와 웹 자원이 그림 8(B)과 같이 상호 작용한다. JMBoard 컨트롤러가 게시글 정보를 획득하기 위한 비즈니스 로직을 수행한다. 그러고 나서 결과 페이지 생성을 위해 jmboard.jsp 뷰로 요청이 포워드된다. 결과 페이지를 생성할 때 여러 페이지가 공유하는 top.jsp (페이지 상단에 “JMBoard ver1.0” 출력)와 bottom.jsp (페이지 하단에 “Copyright...” 출력)를 포함한다. 그리고 이 두 개 웹 컴포넌트 사이에서 이 페이지 특화 내용을 위한 view.jsp(페이지 중간에 게시글 내용, 댓글 작성 UI, 게시글 목록 출력)를 포함한다. 여기까지가 웹 서버에서 일어나는 상호 작용이다.

(A) Page



(B) Page generation graph with resource added

Fig. 7 Page and page generation graph of JBars data for barcode generation

이 후 웹브라우저에서 위 결과 페이지가 로드되는 과정에서 새로운 웹 자원(그림 8(C) 참고)이 식별되고, 각 웹 자원을 필요로 하는 웹 컴포넌트(그림 8(B) 참고) 또한 식별된다. 그림 8(A) 페이지에서 붉은색 박스로 둘러싼 영역이 UI 측면에서 그림 1(A) 페이지와 차이가 나는 부분이다. 이 영역에서 필요한 자원이 그림 8(C)의 20-28번 웹 자원이다. 나머지 1-19번 웹 자원은 그림 8(A) 페이지와 그림 1(A) 페이지가 공유하는 자원이다. 예를 들면 20번 자원(i_delete.gif)은 그림 8(A)에서 주황색 타원으로 표시되어 있고, 붉은색 박스 안에 존재한다. 19번 자원(search.gif)은 파란색 타원으로 표시되어 있다. 그래서 19번 자원을 변경하면, 그림 8(A) 페이지뿐 아니라 그림 1(A) 페이지까지 영향을 받는다는 사실을 파악할 수 있다.

세 번째 앱은 반려 동물을 판매하는 온라인 쇼핑몰

인, JPetStore(https://github.com/mybatis/jpetstore-6)이다. JPetStore는 연구 [17, 18]을 포함한 여러 연구에서 벤치마크로 널리 사용되고 있다. 또한 연구 [19]는 웹 앱의 아키텍처 재공학 기법의 유용성을 보이기 위해 JPetStore를 사례 연구에서 사용하였다.

그림 9는 JPetStore의 메인 페이지를 보여준다. 그림 9(B)를 통해 Catalog.action 컨트롤러가 비즈니스 로직을 수행한 후 Main.jsp 뷰를 통해 결과 페이지를 출력하는 것을 알 수 있다. Main.jsp가 여러 페이지가 공유하는 헤더(페이지 상단에 “JPetStore Demo”, 로그인, 검색 박스 출력)와 푸터(페이지 하단에 “www.mybatis.org" 출력)를 출력한다. 또한 이 두 개 사이에서 이 페이지의 특화 내용(붉은색 박스 안쪽의 내용으로 여러 메뉴를 출력)을 출력한다. JPetStore는 JMBoard와는 달리 헤더와 푸터를 Include로 포함하지는 않는다. 여기까지가 웹 서버에서 일어나는 상호 작용이다.

이 후 웹브라우저에서 위 결과 페이지가 로드되는 과정에서 새 웹 자원(그림 9(C) 참고)이 식별되고, 각 웹 자원을 필요로 하는 웹 컴포넌트(그림 9(B) 참고) 또한 식별된다. 그림 9(A) 페이지에서 붉은색 상자로 둘러싼 영역이 필요로 하는 자원은 그림 9(C)의 10-15번 웹 자원이다. 나머지 1-9번 웹 자원은 다른 페이지와 공유하는 웹 자원이다. 예를 들면 15번 자원(splash.gif)은 그림 9(A)에서 주황색 점선 박스로 표시되어 있고, 메인 페이지에 특화된 웹 자원이다. 반대로 2번 자원(logo-topbar.gif)은 푸른색 점선 박스로 표시되어 있고, 여러 페이지에 등장하는 웹 자원으로 JPetStore 로고를 의미한다.

세 개의 오픈 소스 웹 앱을 이용한 사례 연구를 통해 다음과 같은 사실을 알 수 있다. 첫째, 웹 자원과의 의존 관계를 웹 컴포넌트 수준에서 파악할 수 있다. 그래서 페이지 수준에서 의존 관계를 파악할 때 보다 더 세밀하게 웹 앱을 이해할 수 있다. 둘째, 웹 자원 변경 시 영향 받는 웹 컴포넌트와 웹 페이지를 파악할 수 있다. 셋째, 웹 자원을 공유하는 웹 컴포넌트를 식별할 수 있다. 위와 같이 웹 앱에 대한 이해도가 향상되어 웹 앱의 유지보수가 쉬워질 수 있다.

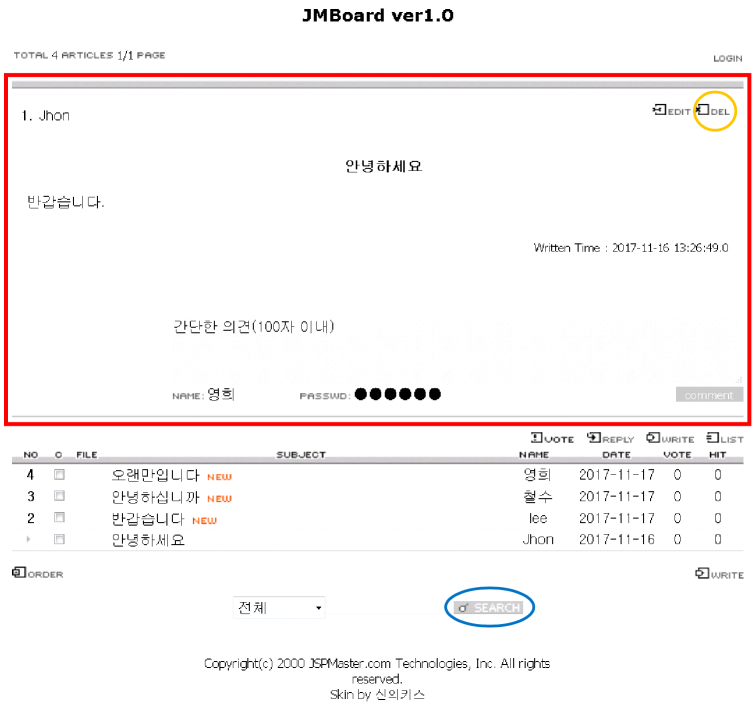
Ⅴ. 결 론

웹 앱의 요구사항이 복잡해지고 빠르게 변화하고 있어, 웹 앱의 유지보수가 더욱더 중요해지고 있다. 따라서 웹 앱의 효과적인 유지보수를 위해, 웹 앱의 이해를 위한 연구가 필요하다. 이를 위해 본 논문은 웹 앱의 페이지 생성 과정을 이해하기 위한 모델과 이 모델을 추출하기 위한 동적 추출 방법을 제시했다. 이 모델과 추출 방법은 기존 연구와 달리 웹 자원과 웹 컴포넌트 사이의 상호 작용까지 고려한다. 그리고 웹 공학 분야의 기존 연구에서 유효성 검증을 위해 사용된, 오픈 소스 웹 앱을 대상으로 실험하여 제안하는 기법의 활용 가능성을 검증했다.

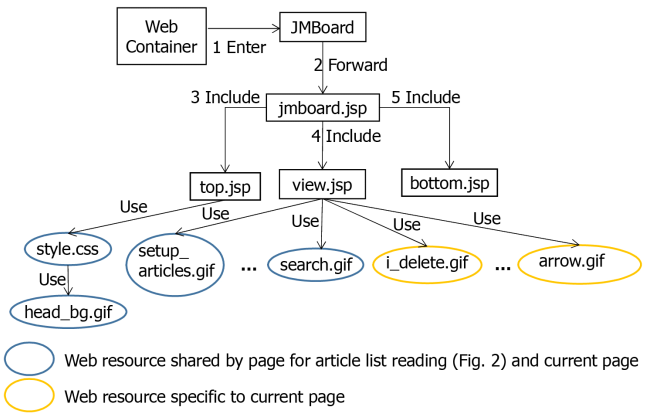
본 논문은 사용자가 페이지를 요청할 때 페이지를 생성하기 위해 필요한 웹 컴포넌트, 웹 자원과 이들 사이의 관계를 동적으로 추출하여 그래프로 표현하는 방법을 제안하였다. 이를 위해 웹 서버에서 서블릿 필터와 래퍼를 활용하여 웹 컴포넌트와 웹 컴포넌트 사이의 상호 작용을 추출하였다. 웹브라우저에서는 파이어폭스의 WebExtension을 활용하여 웹 자원을 식별하고, 웹 컴포넌트와 웹 자원 사이의 사용 관계를 파악하였다.

본 논문이 기여하는 점은 다음과 같다. 첫째, 기존 연구와 달리 웹 서버 측면에서의 실행 컴포넌트(JSP, 서블릿, HTML 등)와 웹 자원(이미지, CSS 파일, JavaScript 파일 등) 사이의 의존 관계를 가시적으로 파악할 수 있는 그래프 모델을 제안한다. 둘째, 자바 웹 앱으로부터 이 그래프 모델을 자동적으로 추출하는 방법을 제안한다. 셋째, 제안하는 역공학 방법은 동적 분석을 활용하기 때문에, 새로운 기술(프로그래밍 언어, 웹 개발 프레임워크 등) 도입에 영향을 덜 받으며 적용 가능하다. 넷째, 오픈 소스 웹 앱에 대한 사례 연구를 통해 제안하는 방법의 활용 가능성을 보였다.

향후 연구로는 페이지 생성 그래프를 효과적으로 가시화하는 방법에 대한 연구를 진행할 수 있다. 둘째, 스크립트 내에서 발생하는 웹 자원 요청(예를 들면 AJAX를 통한 요청)에 대해, 이 요청을 발생하는 웹 컴포넌트를 식별하는 방법에 대한 연구가 필요하다.



(A) Page



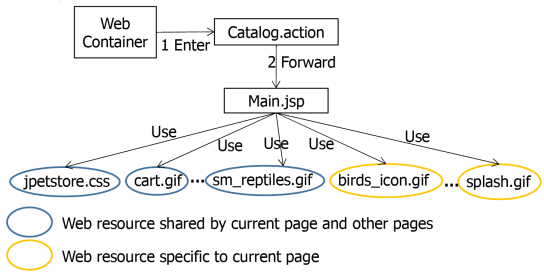
(B) Page generation graph with resource added

No.	Web resource	No.	Web resource
1~19	the same as the web resources shown in Fig 1C	24	i_vote.gif
20	i_delete.gif	25	i_list.gif
21	i_modify.gif	26	icon_passwd.gif
22	icon_homepage.gif	27	icon_name.gif
23	i_reply.gif	28	arrow.gif

(C) Web resources

Fig. 8 Page, page generation graph, and web resources of JMBoard for article reading

(A) Page



(B) Page generation graph with resource added

No.	Web resource	No.	Web resource
1	jpetstore.css	10	birds_icon.gif
2	logo-topbar.gif	11	cats_icon.gif
3	cart.gif	12	dogs_icon.gif
4	separator.gif	13	fish_icon.gif
5	sm_birds.gif	14	reptiles_icon.gif
6	sm_cats.gif	15	splash.gif
7	sm_dogs.gif		
8	sm_fish.gif		
9	sm_reptiles.gif		

(C) Web resources

Fig. 9 Main page, page generation graph, and web resources of JPetStore

ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2017R1D1A1B03029374 and No. 2011-0013781). The present Research has been conducted by the Research Grant of Kwangwoon University in 2018.

References

[1] H. M. Kienle and H. A. Muller, "A WSAD-based fact extractor for J2EE web projects," in *Proceedings of the 9th IEEE International Workshop on Web Site Evolution*, Paris, pp. 57-64, 2007.

[2] I. Zahoor, O. Maqbool, and R. Naseem, "Web application fact extractor (WAFE)," in *Proceedings of the 2013 8th International Conference on Digital Information Management*, Islamabad, pp. 379-384, 2013.

[3] H. M. Kienle and D. Distante, "Evolution of web systems," in *Evolving Software Systems*, 1st ed. Heidelberg: Springer-Verlag Berlin Heidelberg, ch. 7, pp. 201-228, 2014.

[4] A. E. Hassan and R. C. Holt, "Architecture recovery of web applications," in *Proceedings of the 24th International Conference on Software Engineering*, Orlando, pp. 349-359, 2002.

[5] Z. Mushtaq, G. Rasool, and B. Shehzad. (2017, June). Multilingual source code analysis: A systematic literature review. *IEEE Access* [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7953501.

[6] A. Zaidman, N. Matthijssen, M. A. Storey, and A. Van Deursen, "Understanding AJAX applications by connecting client and server-side execution traces," *Empirical Software Engineering*, vol. 18, no. 2, pp. 181-218, Apr. 2013.

[7] J. Oh, W. H. 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, July 2017.

[8] M. Han and C. Hofmeister, "Modeling request routing in web applications," in *Proceedings of the 8th IEEE International Symposium on Web Site Evolution*, Philadelphia, pp. 103-110, 2006.

[9] W. G. Halfond, "Identifying inter-component control flow in web applications," in *Proceedings of the 15th International Conference on Web Engineering*, Rotterdam, pp. 52-70, 2015.

[10] J. Buckley, N. Ali, M. English, J. Rosik, and S. Herold, "Real-time reflexion modelling in architecture reconciliation: A multi case study," *Information and Software Technology*, vol. 61, pp. 107-123, May 2015.

[11] T. Forster, T. Keuler, J. Knodel, and M. C. Becker, "Recovering component dependencies hidden by frameworks--experiences from analyzing OSGi and Qt," in *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, Genova, pp. 295-304, 2013.

[12] J. Oh, W. H. Ahn, and T. Kim, "MVC architecture driven restructuring to achieve client-side web page composition,"

in *Proceedings of the 2016 7th IEEE International Conference on Software Engineering and Service Science*, Beijing, pp. 45-53, 2016.

[13] Oracle. The essentials of filters [Internet], Available: http://www.oracle.com/technetwork/java/filters-137243.html.

[14] MDN Web Docs. Browser extensions [Internet], Available: https://developer.mozilla.org/en-US/Add-ons/WebExtensions.

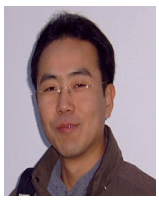
[15] J. Oh, W. H. Ahn, and T. Kim, "MVC architecture-aware restructuring of web apps," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 21, no. 11, pp. 2153-2166, Nov. 2017.

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

[17] Y. Qu, X. Guan, Q. Zheng, T. Liu, J. Zhou, and J. Li, "Calling network: A new method for modeling software runtime behaviors," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 1, pp.1-8, Feb. 2015.

[18] D. Shen, Q. Luo, D. Poshyvanyk, and M. Grechanik, "Automating performance bottleneck detection using search-based application profiling," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, Maryland, pp. 270-281, 2015.

[19] 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.



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

이승현(Seunghyun Lee)

2011년~현재 가톨릭대학교 정보통신전자공학부 재학
※관심분야 : Software Engineering, Web Engineering

김아형(Ah Hyoung Kim)

2015년~현재 가톨릭대학교 컴퓨터정보공학부 재학
※관심분야 : Web Engineering, 알고리즘, 인공지능

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