KIPS Tr. Software and Data Eng.
Vol.6, No.7 pp.329~336 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2017.6.7.329

Automatic Extraction of Component Collaboration in Java Web Applications by Using Servlet Filters and Wrappers

Jaewon Oh† · Woo Hyun Ahn†† · Taegong Kim†††

ABSTRACT

As web apps have evolved faster and become more complex, their validation and verification have become essential for their development and maintenance. Efficient validation and verification require understanding of how web components collaborate with each other to meet user requests. Thus, this paper proposes a new approach to automatically extracting such collaboration when a user issues a request for a new page. The approach is dynamic and less sensitive to web development languages and technologies, compared to static extraction approaches. It considers an orignal web app as a black-box and does not change the app's behavior. The empirical evaluation shows that our approach can be applicable to extract component collaboration and understand the behavior of open source web apps.

Keywords : Web Component Collaboration, Filter and Wrapper, Servlet

자바 웹 앱에서 서블릿 필터와 래퍼를 이용한 컴포넌트 협력 과정 자동 추출 기법

오 재 원† · 안 우 현†† · 김 태 공†††

요 약

웹 앱은 빠르게 진화하며 나날이 복잡해지고 있다. 이에 따라 웹 앱의 검증(validation & verification)이 웹 앱의 개발 및 유지보수를 위해 더욱 중요해 지고 있다. 효율적인 검증을 위해서는 웹 앱 실행 시 일어나는 웹 컴포넌트 사이의 협력 과정(collaboration)에 대한 이해가 선행되어야 한다. 이를 위해 본 논문에서는 웹 페이지 요청 시 페이지 생성을 위해 실행되는 웹 컴포넌트와 이들의 협력 과정을 자동적으로 추출하는 기법을 제안한다. 제안하는 기법은 복잡한 웹 개발 언어 및 기술에 덜 의존하는 동적 기법이며 웹 앱 소스를 블랙박스로 보고 웹 앱의 기능 변화 없이 협력 과정을 추출한다. 오픈 소스 웹 앱에 적용하는 실험을 통해 제안하는 기법의 유효성을 확인할 수 있다.

키워드 : 웹 컴포넌트 협력 과정, 필터와 래퍼, 서블릿

1. 서 론

웹 앱 사용자는 하이퍼링크 클릭 혹은 폼 제출을 통해 새로운 페이지를 요청한다. 이러한 요청 시 웹 서버에서는 여러 웹 컴포넌트(서블릿, JSP, HTML 등)가 협력하여 새 페이지를 생성한다. 빠르게 진화하고 복잡해지는 웹 앱의 요구사항 검증과 유지보수를 위해서 이러한 웹 컴포넌트 협력 과정(collaboration)을 이해하고 문서화하는 과정이 필요하다. 위 협력 과정을 연구 [1]에서는 요청 처리 경로(request route)라고 정의한다. 연구 [2]에서는 컴포넌트간 제어 흐름(inter-component control flow)라는 개념을 유사한 용도로 사용한다. 앞으로 웹 컴포넌트 협력 과정과 요청 처리 경로를 같은 의미로 사용한다.

Fig. 1에서 웹 컴포넌트 협력 과정의 한 예를 볼 수 있다. 페이지 요청을 웹 컨테이너가 받고 요청을 처리할 웹 컴포넌트 C를 식별한 후 실제 요청 처리가 시작된다(단계 1). C가 비즈니스 로직을 처리한 후 컴포넌트 T에게 응답 메시지 생성을 요청한다(단계 2). T는 P의 수행 결과를 포함(단계 3)하고 Q의 수행 결과를 포함(단계 4)하여 응답 메시지를 생성한다. 그리고 이 응답 메시지는 웹 컨테이너로 전달된다. 웹 컨테이너는 최종적으로 브라우저에게 이 응답을 전달한다. 이러한 행동 모델은 순공학 시 웹 컴포넌트 구현 전 설계되고 구현 시 활용된다.

※ 이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2017R1D1A1B03029374). 본 연구는 2017년도 가톨릭대학교 교비연구비의 지원으로 이루어졌음. 이 논문은 2016년도 광운대학교 교내 학술연구비 지원에 의해 연구되었음. 실험을 위해 도움을 준 가톨릭대학교 컴퓨터정보공학부 최윤호 학부생에게 고마움을 전함.

† 종신회원 : 가톨릭대학교 컴퓨터정보공학부 부교수
†† 정 회 원 : 광운대학교 컴퓨터소프트웨어학과 교수
††† 정 회 원 : 인제대학교 컴퓨터공학부 부교수

Manuscript Received : December 12, 2016
First Revision : February 8, 2017
Accepted : February 15, 2017

* Corresponding Author : Woo Hyun Ahn(whahn@kw.ac.kr)

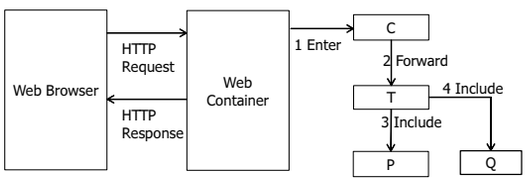


Fig. 1. Example of Component Collaboration in Java Web App

웹 앱으로부터 요청 처리 경로를 자동적으로 추출하는 단계는 다음과 같은 이유로 역공학이나 순공학을 위해 중요한 과정이다. 첫째, 웹 앱은 빠르고 지속적으로 진화해야 하는 요구가 있다. 그래서 빠른 출시를 위해 설계 과정 없이 소스 코드 위주로만 개발이 이루어지는 경우가 적지 않다. 설계 부재로 발생하는 유지보수 어려움을 해결하기 위해 역공학 단계에서 요청 처리 경로가 자동 추출되면 좋을 것이다. 둘째, 순공학 시 개념적 모델(아키텍트가 설계한 모델)에 맞게 구현이 되었는지 검증하는 단계가 중요하다. 이 단계에서 요청 경로 처리를 추출하면 개념적 모델과 실제 웹 소스 코드 사이의 일치성 검사[3]에 도움이 된다. 셋째, 자바 웹 기술은 JSP/서블릿 기술, expression language, JSTL 등의 태그 라이브러리, 배포 서술자(web.xml), 자바 어노테이션 등 여러 요소로 구성되며, 이들 요소는 요청 처리 경로 결정에 영향을 준다. 따라서 수작업으로 요청 처리 경로를 식별하려면 이들 요소의 규칙을 종합적으로 고려해야 하기 때문에 상당한 비용이 요구된다.

기존 자동 추출 방법은 크게 정적 분석[1-2]과 동적 분석[4-5]으로 나눌 수 있다. 정적 분석은 웹 앱 관련 소스 코드, 문서 등을 파싱하고 분석하여 요청 처리 경로를 추출한다. 둘째, 동적 분석은 웹 앱을 실행하여 실행 로그 혹은 트레이스를 얻고 이것을 분석하여 경로를 추출한다. 동적 분석은 웹 앱 구현 언어, 라이브러리, 플랫폼 등에 독립적인 장점을 지니고 있다. 그러나 가능한 모든 실행 흐름을 얻는 것이 목적이라면, 동적 분석의 경우 분석을 위해 고려하는 시나리오가 웹 앱의 가능한 실행 시나리오를 얼마나 커버하는 지가 중요한 이슈가 된다. 정적 분석은 동적 분석의 장점을 단점으로 단점을 장점으로 지닌다. 또한 [5]에서 지적하듯이 여러 프로그래밍 언어를 함께 사용하는 웹 앱의 경우 정적 분석만으로 웹 앱의 설계를 추출하는 것이 어렵다.

기존 주요 관련 연구가 지닌 문제는 다음과 같다. [4]는 서블릿 필터(Filter)[6]를 사용하여 요청 처리 경로의 추출이 가능함을 언급하였다. 그렇지만 구체적인 추출 방법이 제시되지 않았고 래퍼(Wrapper)[6] 없이 필터만으로는 추출이 어렵다. [2]는 자바 웹 앱의 핵심 기술인 web.xml에 대한 언급이 없고, 서블릿/JSP간의 상호작용 중 하나인 forward 액션에 대한 고려도 없다. [1]은 동적으로 생성되는 요청 URL에 대한 언급이 없다.

본 논문은 위 문제 해결 위해 자바 웹 앱으로부터 서블릿 필터와 래퍼를 사용하여 요청 처리 경로를 추출하는 동적 방법을 제안한다. 복잡한 웹 개발 언어/기술에 덜 의존하는 동적 기법이며 웹 앱 소스를 블랙박스로 보고 서블릿 필터와 래퍼만을 플러그인 하여 요청 처리 경로를 웹 앱의 기능 변화 없이 추출한다. 아울러 expression language 등을 사용하여 동적으로 생성되는 URL을 인식할 수 있으며, 자바 웹 앱 컴포넌트 사이의 모든 상호작용(forward, include, redirect 액션)을 고려한다. 둘째 기존 연구와 달리 추출한 실재적 모델이 개념적 모델과 일치하는지 효과적으로 점검[3] 위해 문자열 형태로 이 행동 모델을 표현하는 방법을 제안한다.

2. 필터, 래퍼 이용한 요청 처리 경로 자동 추출

2.1 배경

우선 제안하는 기법의 이해를 위해 서블릿 필터와 래퍼[6]에 대해 설명한다(Fig. 2 참고). 페이지 요청 시 웹 서버 측에서 여러 웹 컴포넌트가 협력하여 요청을 처리한다. 웹 컴포넌트는 입력을 읽고 결과를 작성하기 위해 요청 메시지 객체(Request object)와 응답 메시지 객체(Response object)를 사용한다. 때로는 웹 앱 내에서 이 메시지 객체들을 포장할 필요가 있으며(예를 들면, 텍스트 인코딩 변환 등의 작업을 위해) 이 경우 래퍼 객체(Request wrapper object, Response wrapper object)를 사용할 수 있다. 필터는 웹 컴포넌트 실행 전후 사전(Pre) 작업, 사후(Post) 작업 수행 위해 사용할 수 있는 객체다. 예를 들면, Fig. 1의 각 단계마다 필터를 사용할 수 있다.

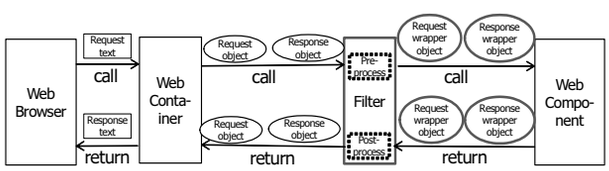


Fig. 2. Filter and Wrapper

예를 들면 원화를 처리할 수 있는 웹 앱이 있다고 하자. 그런데 사용자가 미화를 입력하더라도 동작하도록 확장하고 싶다. 이 경우 페이지 요청이 발생하면 첫 번째 웹 컴포넌트(F라 하자)가 수행되기 전에 입력 메시지 객체를 스캔하여 달러 단위의 숫자를 원화로 변경하면 좋을 것이다. 즉 필터 객체가 컴포넌트 F 수행 이전에 실행되어 사전 작업으로 이러한 변경 작업을 진행할 수 있다. 그리고 모든 웹 컴포넌트들이 원화로 작업할 수 있도록 원본 입력 메시지 객체 대신 원화로 변경된 입력 메시지 래퍼 객체를 사용하게 한다.

2.2 요청 처리 경로 모델 정의

페이지 요청 한 개를 처리하기 위한 요청 처리 경로(collaboration, request route)를 표현하기 위해 아래 (정의 1)에서 그래프 모델을 제안한다. Fig. 1에서 웹 컨테이너를 포함하여 웹 컨테이너 우측에 그려진 그래프가 이 경로 모델의 한 인스턴스이다.

(정의 1). (요청 처리 경로 그래프) 페이지 요청 r이 주어질 때, r의 요청 처리 경로 그래프 $G_r(V, E, t, o)$는 방향

그래프이며 다음 조건을 만족한다.

- V: r의 요청 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. 단, 네 가지 호출 타입(*Enter, Forward, Include, Redirect*)이 존재하며[9], 아래에서 구체적으로 설명한다.
- t: $E \rightarrow$ {*Enter, Forward, Include, Redirect*}. 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 forward 액션으로 호출하면 $t(e_i)$=*Forward*가 된다. 가능한 호출 타입은 다음과 같다.
  - *Enter*: 요청 처리 시 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(include 되는) 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 타입과 일관된 정의를 위해서이다.
- o: $E \rightarrow \{1, 2, ..., |E|\}$. o는 E에 속하는 각 간선의 호출 순서를 반환하는 함수이다. 즉, 요청 처리 경로를 구성하는 컴포넌트의 수행 순서 정보를 표현한다. 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.

2.3 동적 분석 통한 요청 처리 경로 추출

추출을 위한 기본 요구사항은 다음과 같다. 첫째, 요청 처리 과정에서 실행되는 모든 웹 컴포넌트를 식별해야 한다. 둘째, 웹 컴포넌트 실행 순서대로 컴포넌트 ID(혹은 URL)와 이 웹 컴포넌트가 실행된 이유(즉, 호출 종류)를 파악하고 기록해야 한다. 예를 들면 “Forward C”는 앞선 웹 컴포넌트가 컴포넌트 C에게 forward 요청을 호출한다는 것을 의미한다. 셋째, 웹 컴포넌트 실행 시작과 종료 시 이 사실을 기록해야 한다. 기록하는 이유는 웹 컴포넌트 호출도 일반 프로그래밍 언어에서의 함수 호출처럼 중첩이 가능하므로 실행 순서에 관한 애매모호함을 제거하기 위해서이다.

제안하는 요청 처리 경로 추출 알고리즘이 Fig. 3에 나와 있다. 이 알고리즘은 최종적으로 (정의 1)에서 정의한 요청 처리 경로 그래프와 이에 대응하는 경로 문자열을 생성한다.

첫째, 요청 처리 경로 그래프 추출의 일반 규칙은 각 웹 컴포넌트 c 마다 필터를 두고 c가 호출되면, c가 실행되기 직전에 c의 사전 필터(pre-filter)가 c의 ID, 호출 관계(<caller, c>), 호출 타입, 호출 순서를 식별하고 그래프 구성 요소에 추가한다는 것이다.

집합 V는 다음과 같이 구할 수 있다. 이 과정에서 중요한 일은 웹 컴포넌트 ID를 식별하는 방법이다. 앞으로 호출 타입이 *Enter*, *Forward*, *Include*인 간선에 부착된 필터를 차례대로 Enter 필터, Forward 필터, Include 필터라 하자. Enter 필터와 Forward 필터의 경우는 사전 필터 작업 시 Java API(HttpServletRequest의 getServletPath() 등)를 이용하면 사전 필터 작업 후 실행할 웹 컴포넌트 ID를 식별할 수 있다(Fig. 3 줄 번호 8-9, 14-15).

Include 필터의 경우는 위 방법이 가능하지 않다. 왜냐하면, 위 API를 사용하면 include되는 컴포넌트의 ID가 아니라 include하는 웹 컴포넌트의 ID가 반환되기 때문이다. 그래서 입력 메시지 래퍼 객체의 도움을 받아 ID 식별한다(Fig. 3 줄 번호 11, 18-19). 즉 $<v_i, v_j>$가 include 호출이라고 하자. 그러면 v_i의 소스코드에 include 호출이 존재하고, 이 호출 직전에 include될 대상을 지정하는 Java API (ServletRequest의 getRequestDispatcher()) 호출 코드가 존재한다. 본 연구에서는 이 메서드를 재정의하여 include로 실행될 컴포넌트 ID를 식별한다(Fig. 3 줄 번호 19, 27-34). 이렇게 새롭게 만든 클래스가 IncludeRequestWrapper이며, 이 클래스의 객체(reqWrapper)를 요청 처리 경로가 시작할 때, 즉 Enter의 사전 필터 작업 시 생성하여 기본 요청 메시지 객체 대신 사용한다(Fig. 3 줄 번호 11).

redirect는 브라우저로 하여금 새롭게 알려주는 URL로 재접속하라고 요청하는 메시지로 브라우저가 메시지 수신자여서 수신 웹 컴포넌트가 없어 필터를 추가할 수 없다. 그래서 redirect를 명령하는 JAVA API(ServletResponse의 sendRedirect())를 응답 메시지 래퍼에서 재정의하여 redirect URL을 식별한다(Fig. 3 줄 번호 12, 35-44). 즉 $<v_i, v_j>$가 redirect 호출이라고 하자. 그러면 v_i의 소스코드에 sendRedirect(v_j) 호출이 존재하며, 이 코드가 실행되면 재정의한, 응답 메시지 래퍼 객체의 sendRedirect()가 호출된다.

집합 E를 구하는 방법은 다음과 같다. 요청이 웹 서버에 도착하면, 웹 컨테이너를 첫 컴포넌트로 하여 요청이 호출 순서를 따라 웹 컴포넌트를 옮겨가며 처리가 된다. 현재 실행되는 컴포넌트를 v_{cur}라 하자(Fig. 3 줄 번호 3). 제일 처음 실행되는 컴포넌트는 웹 컨테이너이며, 두 번째 컴포넌트는 Enter로 웹 컨테이너와 연결된 컴포넌트이다. 그 이후는 웹 앱 소스코드를 따르게 된다. E를 구하는 기본 원리는 사전 필터가 실행할 때마다 호출될 컴포넌트와 함께 간선 정보를 그래프에 넣는다(Fig. 3 줄 번호 10, 16, 20, 39, 45-46, 50-55). 그리고 제어 흐름이 호출된 컴포넌트로 옮겨가기 때문에, v_{cur}를 다음 실행할 v_{next}로 변경한다(Fig. 3 줄 번호 48). 반대로 사후 필터에서는 v_{cur}가 막 수행을 마치고, 제어 흐름이 v_{cur}를 호출한 웹 컴포넌트로 옮겨가기 때문에 v_{cur}를 v_{cur}의 호출자로 변경한다(Fig. 3 줄 번호 24). redirect의 경우는 필터가 없기 때문에, 위 일을 응답 메시지 래퍼에서 수행한다(Fig. 3 줄 번호 35-44).

t 함수는 사전 필터에서 구할 수 있다(Fig. 3 줄 번호 10, 16, 20). 예를 들면, Forward 필터가 호출되면 해당 웹 컴포넌트가 forward 액션에 의해 호출된다는 의미이다. redirect의 경우는

응답 메시지 래퍼 객체에서 식별한다(Fig. 3 줄 번호 39).

o 함수는 간선을 호출 순서대로 순서화하면 된다. 그래서 현재까지 발생한 호출 수(Fig. 3 줄 번호 2)를 0으로 초기화하고, 호출이 발생할 때마다 1씩 증가시키며, 현재 값을 지금 발생한 호출의 순서 값으로 활용한다(Fig. 3 줄 번호 54).

둘째, 요청 처리 경로를 표현하는 문자열을 추출하는 일반 규칙은 다음과 같다. 필터 실행 순서대로 요청 처리 경로 문자열(Fig. 3 줄 번호 4)에 컴포넌트 ID와 호출 타입을 덧붙인다. 예를 들면 컴포넌트 C가 Forward로 실행될 경우(Fig. 3 줄 번호 16, 47, 56-58), C 수행 직전에 C의 사전 필터가 C가 수행을 시작한다는 표시("(")를 하고 C의 호출 타입과 C의 ID를 기록한다. 마지막으로 C와 C가 호출한 컴포넌트가 수행을 모두 마치면(Fig. 3 줄 번호 22-23) C의 사후 필터가 C의 처리가 끝났다는 표시(")")를 덧붙인다(Fig. 3 줄 번호 59-61). 즉, 호출이 중첩될 수 있으므로 괄호를 이용해 중첩 관계를 명시한다. redirect의 경우는 응답 메시지 래퍼 객체에서 위 일을 수행한다(Fig. 3 줄 번호 39-40). Table 1은 유효한 요청 처리 경로 문자열 몇 개를 보여준다.

Fig. 4는 위 문자열 형태의 요청 처리 경로 생성 위해 필터와 래퍼가 어떤 수행 과정을 거치는지 설명한다. 왼쪽에

```
Find Gr(V, E, t, o) and its collaboration string for a request r
 1: let V={}, E={}, t={}, o={} //initialize the sets to an empty set
 2: initialize nOfCalls to zero //nOfCalls is the total number of calls known until now
 3: initialize v_cur to a web container //v_cur is a web component currently running in r's collaboration
 4: initialize collabString to "" //collabString is a string corresponding to r's collaboration
 5: set CALL_START to "(" //CALL_START is a constant string representing the start of a call
 6: set CALL_END to ")" //CALL_END is a constant string representing the end of a call
 7: //the start of core actions that the filters perform
 8: if (r is routed to a pre-filter of Enter call) {
 9:    identify the callee v_next using Java API such as getServletPath() & getQueryString() of HttpServletRequest
10:    addNewComponent(v_next, "Enter")
11:    construct a IncludeRequestWrapper instance reqWrapper used instead of the default request object
12:    construct a RedirectResponseWrapper instance used instead of the default response object
13: }
14: else if (r is routed to a pre-filter of Forward call) {
15:    identify the callee v_next as in line 9
16:    addNewComponent(v_next, "Forward")
17: }
18: else if (r is routed to a pre-filter of Include call) {
19:    v_next = reqWrapper.includedCompURL
20:    addNewComponent(v_next, "Include")
21: }
22: else if (r is routed to a post-filter of Enter, Forward, or Include call) {
23:    addEndOfCallToCollabString()
24:    v_cur = caller of v_cur
25: }
26: // the end of core actions that the filters perform
27: //define the request wrapper class to identify the URL of the included component
28: class IncludeRequestWrapper extends HttpServletRequestWrapper {
29:    includedCompURL //to store the URL of the included component
30:    getRequestDispatcher(url) {
31:       includedCompURL = url
32:       super.getRequestDispatcher(url)
33:    }
34: }
35: //define the response wrapper class to perform actions for the Redirect call
36: class RedirectResponseWrapper extends HttpServletResponseWrapper {
37:    sendRedirect(destinationUrl) {
38:       v_next = destinationUrl
39:       addNewComponent(v_next, "Redirect")
40:       addEndOfCallToCollabString()
41:       v_cur = caller of v_cur
42:       super.sendRedirect(destinationUrl)
43:    }
44: }
45: addNewComponent(v_next, typeOfCall) {
46:    addComponentToCollabGraph(v_next, typeOfCall)
47:    addStartOfCallToCollabString(v_next, typeOfCall)
48:    v_cur = v_next
49: }
50: addComponentToCollabGraph(v_next, typeOfCall) {
51:    V = V ∪ {v_next}
52:    E = E ∪ {<v_cur,v_next>}
53:    t = t ∪ {(<v_cur,v_next>, typeOfCall)}
54:    o = o ∪ {(<v_cur,v_next>, ++nOfCalls)}
55: }
56: addStartOfCallToCollabString(v_next, typeOfCall) {
57:    collabString += CALL_START + typeOfCall + v_next
58: }
59: addEndOfCallToCollabString() {
60:    collabString += CALL_END
61: }
```

Fig. 3. Extraction of Request Route Using Filter and Wrapper

Table 1. Valid Request Routes

Request route	Description
(Enter C)	When a request is made to a web server, the request is routed to web component C, which is executed as the first component of the current collaboration. Finally, when C finishes its job, a response is returned to a browser.
(Enter C (Include D))	When a request is made to a web server, web component C is executed as the first component of the current collaboration. In the middle of execution of C, web component D is included into C, which continues its execution. When C finishes its job, a response is returned to a browser.
(Enter C (Include D) (Include E))	When a request is made to a web server, web component C is executed. In the middle of execution of C, web components D and then E are included into C. Finally, C finishes its job and a response is returned to a browser.
(Enter C (Include D (Include E)))	When a request is made to a web server, web component C is executed. In the middle of execution of C, C includes web component D, which also includes web component E. E, D, and C finish their jobs in order and then a response is returned to a browser.
(Enter C (Forward T (Include P) (Include Q)))	as same as the collaboration shown in Fig 1

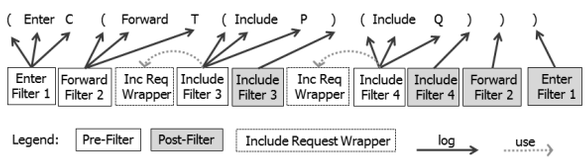


Fig. 4. Filter and Wrapper for Request Route Extraction

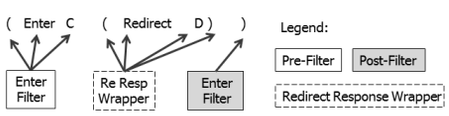


Fig. 5. Redirect Request Route Extraction

서부터 차례로 경로 문자열에 필터가 수행 정보를 추가한다. 아울러 사전 필터 작업과 사후 필터 작업이 구분되어 있다. 두 번째 예제로 redirect를 사용할 때 일어나는 과정을 보도록 하자. Fig. 5는 요청을 통해 웹 컴포넌트 C가 수행 도중 브라우저로 하여금 URL D로 재접속하도록 요청하는 것을 표현한다. Table 2는 역공학을 위해 설계한 필터와 래퍼를 설명한다.

추출한 요청 처리 경로는 아키텍처 적합성 검사를 위해 사용할 수 있다. 즉, 아키텍트가 설계한 개념적 모델과 위에서 제시한, 역공학을 통해 만들어낸 실재적 모델이 일치하는지 점검할 수 있다. 이러한 점검의 기반을 위해 (정의 1)의 요청 처리 경로 그래프를 Table 1과 같은 문자열 형태로 매핑하면, 일치성 검사를 위해 기존의 문자열/패턴 매칭 알고리즘을 이용할 수 있다.

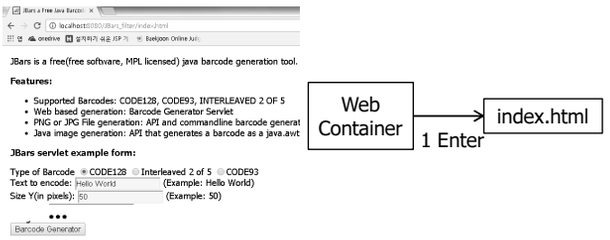
Table 2. Filter and Wrapper Classes

Filter / Wrapper (Parent class/Interface)	Description
EnterFilter (Filter)	pre-processing: log the arrival of a request on a server, an ID of the first web component of the current collaboration, and "Enter" post-processing: log the termination of processing of the current request
ForwardFilter (Filter)	pre-processing: log the start of a Forward call, an ID of a web component that receives the Forward call, and "Forward" post-processing: log the end of the Forward call
IncludeFilter (Filter)	pre-processing: log the start of a Include call, an ID of a web component that receives the Include call, and "Include" post-processing: log the end of the Include call
IncludeRequestWrapper (HttpServletRequestWrapper)	identify an ID of a web component that receives an Include call
RedirectResponseWrapper (HttpServletResponseWrapper)	log the start of a Redirect call, a URL that a browser will be redirected to, "Redirect", and the end of the call

3. 실험 및 결과 분석

유효성 검증을 위해 세 개의 오픈 소스 웹 앱(JBars, JMBoard, Online Movie Ticket Booking)을 대상으로 실험하였다. 이 웹 앱을 선택한 이유는 유명 오픈 소스 저장소에 코드가 공개되어 있고, HTML, JavaScript, 서블릿, JSP, 표준 액션 태그, AJAX 등으로 구성되어 있어 Java 웹 앱의 전형적인 구성을 따르기 때문이다.

JBars(https://sourceforge.net/projects/jbars/?source=directory)는 바코드 생성 웹 앱이다. 이 앱의 전형적인 페이지와 해당 페이지 요청 시 발생하는 요청 처리 경로(그래프와 대응하는 문자열)가 Fig. 6에 나와 있다. Fig. 6A 페이지에서 정보 입력 후 생성 버튼(Barcode Generator)을 누르면 생성된 바코드(Fig. 6B)를 볼 수 있다. (Fig. 6A) 페이지를 위한 요청 처리 경로는 (Enter /index.html)로 뷰(바코드 생성 위한 입력 UI 프레젠테이션) 역할을 하는 index.html 컴포넌트만으로 요청이 처리된다. (Fig. 6B) 페이지를 위한 요청 처리 경로는(Enter /BarcodeServlet)로 BarcodeServlet 컴포넌트가 컨트롤러(바코드 생성)와 뷰(생성된 바코드 프레젠테이션) 역할을 동시에 수행한다. 즉 웹 앱의 권장 아키텍처인



(A) Input page for barcode generation

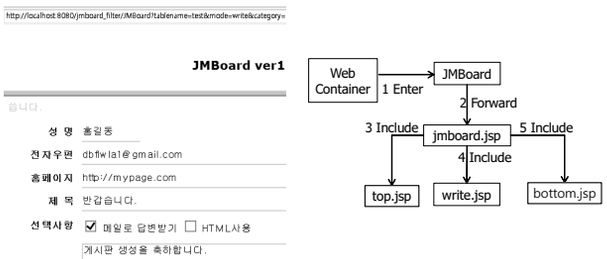
(B) Page for showing barcode

Fig. 6. Typical Pages and Request Routes of JBars

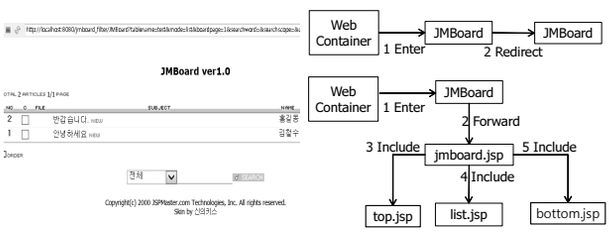
MVC(Model-View-Controller) 구조를 따르고 있지 않다는 사실을 파악할 수 있다. 우리는 이러한 구조적 취약점을 아키텍트에게 알려 주어 수정할 수 있게끔 도와 줄 수 있다.

JMBoard(http://happycgi.com/8168)는 게시판을 생성/삭제할 수 있으며, 생성된 게시판에 게시글을 올릴 수 있게 하는 웹 앱이다. 전형적인 요청인 게시글 작성에 대해 요청 처리 경로를 추출한 결과가 Fig. 7에 나와 있다. Fig. 7A의 게시글 작성을 위한 입력 UI 페이지를 생성하기 위해 관련 컴포넌트가 다음과 같이 상호작용한다. JMBoard 컨트롤러가 필요한 비즈니스 로직을 수행한 후 jmboard.jsp 뷰로 요청이 포워딩되며 이 뷰가 결과 페이지를 생성한다. 생성 시 여러 페이지가 공유하는 top.jsp(페이지 상단에 “JMBoard ver1.0” 출력), bottom.jsp 컴포넌트(페이지 하단에 “Copyright...” 출력)를 포함하고, 이 페이지 특화 내용을 위한 write.jsp(페이지 중간에 성명, 전자우편 등의 입력 UI 출력)를 포함한다. (Fig. 7B) 페이지를 위한 요청 처리 경로는 다음과 같다. JMBoard 컨트롤러(게시글 정보 저장)를 수행한 후 double submit 문제[7]를 방지하기 위해 새로운 URL로 redirect한다. redirect 후 Fig. 7A와 유사한 방식으로 처리된다. 이 경우 페이지 특화 내용을 위해서 list.jsp(게시글 리스트 출력)가 사용된다. 이 웹 앱은 MVC 구조를 따르며 특히 double submit 문제 해결을 위해 post-redirect-get 패턴[7]을 사용하고 있음을 확인할 수 있다. 이와 같은 요청 처리 경로 식별은 [1, 2, 4, 5, 8]에서 언급하듯이 웹 앱을 재구조화(예를 들면, 전통적인 웹 앱을 AJAX 웹 앱으로 재구조화[8])하는데 도움을 줄 수 있다.

세 번째, 영화 예매 웹인 Online Movie Ticket Booking (https://sourceforge.net/projects/onlinemovietick/?source=directory)은 페이지 전환 없이 수행되는 단일 페이지 앱(single-page app)이다. 로그인(Log in 버튼) 메뉴를 선택한 결과가 Fig. 8A에 나와 있다. 사용자 요청의 처리 결과가 나오는 영역을 붉은색 상자로 표시했다. 이 경우 요청 처리 경로는(Enter /login.html)이며, AJAX 호출 통해 얻은, id와 password를

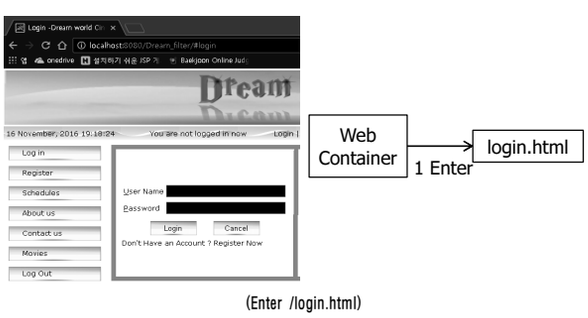


(A) Input page for posting

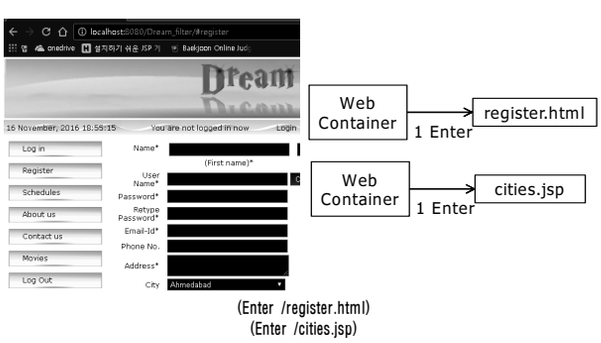


(B) Result Page of posting

Fig. 7. Typical Pages and Request Routes of JMBoard



(A) Input page for logging in



(B) Input page for user registration

Fig. 8. Typical Pages and Request Routes of Ticket Booking

입력하는 화면(login.html)이 붉은색 상자 안에 나와 있다. 이 때 AJAX 호출 여부는 크롬 인스펙터 등을 통해 알 수 있다.

한편, 회원 등록(Register 버튼) 메뉴를 선택한 결과가

Fig. 8B에 나와 있다. 이 페이지의 요청 처리 경로는 (Enter /register.html), (Enter /cities.jsp)으로 두 개의 웹 컴포넌트(register.html, cities.jsp)를 AJAX 호출 통해 접근하는 것이다. 그런데 이 경로는 두 가지로 해석 가능하다는 점에서 애매하다. 즉, 제안하는 기법으로는 Register 버튼 클릭 핸들러에서 regisiter.html, cities.jsp를 모두 호출했는지 아니면 핸들러가 register.html를 호출하고 이 html이 cities.jsp를 호출했는지 파악할 수 없다. 그렇지만 이 경우 정적 분석을 제안하는 동적 분석과 결합하게 되면 이러한 애매함을 제거할 수 있다. 즉, 관련 문서인 Register 버튼 클릭 핸들러 소스코드와 register.html 소스코드의 리뷰를 통해서 두 가지 가능한 해석 중 후자임을 확인할 수 있다.

Fig. 6, Fig. 7, Fig. 8에서 볼 수 있는 URL은 Java, JavaScript, JSTL, Expression language 등을 통해 동적으로 생성되는 URL을 포함한다. 우리 기법은 Fig. 3에서 알 수 있듯이 자바 API를 적절히 활용하여 동적으로 생성된 URL을 정확히 식별하여 요청 처리 경로를 추출할 수 있다.

요청 처리 경로를 추출할 때 본 논문은 다음을 가정한다. 하나의 요청을 처리하는 도중에 다른 요청이 발생하지 않는다. 그래서 Fig. 8B를 통해 언급한 것처럼 한 개의 사용자 요청에 대해 두 개의 요청 처리 경로 문자열이(redirect로 연결되지 않고) 발생한 경우는 한 개의 요청이 끝난 후 다른 요청이 수행되었는지, 두 개가 동시에 처리되었는지 본 논문의 방법으로는 판단할 수 없다. 그래서 이 경우 정적 분석과의 결합을 통해 해결할 필요가 있다.

마지막으로 추출한 요청 처리 경로가 올바른지 검증하는 방법이 필요하다. 이를 위해 [1, 2]와 같은 정적 분석 기법이나 코드 리뷰 등을 활용할 수 있다.

4. 관련 연구

웹 앱의 동작을 이해하기 위해 많은 연구들이 수행되어 왔다. 이 연구를 웹의 아키텍처인 클라이언트-서버 구조를 고려하여 두 가지 방향으로 나누어 살펴볼 수 있다. 즉, 클라이언트(웹브라우저) 관점과 서버 관점에서 웹 앱을 분석할 수 있다. 서버 스크립트(JSP, PHP 등)로 작성된 앱을 분석할 때 서버 스크립트가 아니라 최종적으로 클라이언트에서 실행되는 HTML 페이지를 분석의 대상으로 삼을 수 있다. 이런 경우가 클라이언트 관점에서의 분석이다. 반대로 서버 스크립트를 분석 대상으로 삼으면 서버 관점의 분석이 된다.

클라이언트 관점에서 웹 앱의 동작을 분석하는 최근 경향은 자바스크립트 앱의 증가를 반영하여 주로 AJAX 앱을 대상으로 한다. [10, 11]은 AJAX 앱을 동적 분석하여 UI 관점에서의 상태 변화를 식별하고 앱의 동작을 표현하는 상태 모델을 추출한다. HTML 페이지 관점에서 분석하기 때문에 클라이언트 측면에서의 동작을 이해하는데 도움을 주지만, 본 연구와 달리 서버에서 일어나는 동작에 대해서는 정보를 제공하지 않는다. [12, 13]는 [10]에 기반을 두고 AJAX 앱의 검증을 위한 뮤테이션 테스팅, 테스트 케이스 생성 방법을 제안한다. [11]에는 클라이언트 관점에서 웹 앱을 동적 분석하는 기존 연구가 잘 정리되어 있다.

[14]는 [10-11]과 달리 AJAX 앱을 정적 분석하여 클라이언트 측면에서 앱의 동작을 이해하는데 도움을 주는 제어 흐름 그래프와 요청 그래프를 생성하는 방법을 제안한다. 이 연구의 목적은 개발자가 의도하지 않는 순서로 사용자가 웹 서버의 페이지/서비스를 요청하여 생기는 오류를 막는 것이다. 개발자가 의도하는 모델(요청 그래프)을 따라 사용자가 요청을 하는지 점검하는 프락시를 서버와 클라이언트 사이에 두어, 이 모델을 따르지 않는 요청이 발생하지 않도록 한다. 요청 그래프의 노드는 서버로의 요청이나 서버로부터의 응답을 표현하며, 간선은 실행 순서를 의미한다. [14]는 본 연구와 달리 서버에서의 처리 과정은 고려하지 않는다.

서버 관점에서 분석하는 기존 연구로 [15-16, 5]가 있다. [15]는 웹 앱에서 기대하지 않는 페이지 내비게이션 시퀀스가 오류를 발생시키는 경우가 많다는 점에 주목한다. 그래서 내비게이션 제약 조건을 개발자가 상태 기계 형태로 명세하는 방법과, 이 제약 조건을 따르도록 웹 앱의 제어 흐름을 강제하는 런타임을 제안한다. 아울러 상태 기계를 웹 앱 실행 전에 정적으로 검증하기 위해 모델 체킹을 활용하는 방법을 제안한다. 상태 기계의 노드는 영속적 데이터와 세션의 상태 조합이며, 간선은 사용자 요청을 표현한다. 이 연구에서 주목하는 내비게이션 시퀀스는 페이지 요청을 원소로 한다. [5]는 PHP 웹 앱을 동적 분석하여 페이지 내비게이션 모델을 표현하는 시퀀스 다이어그램을 추출한다. 시퀀스 다이어그램에서 표현하는 객체를 이 연구에서는 서버 페이지로 보며, 서버 페이지 사이의 전이를 메시지로 표현한다. [16]은 Struts 기반 웹 앱을 정적 분석하여 [5]와 유사한 형태의 페이지 내비게이션 모델 추출한다. 따라서 우리 연구는 페이지 요청 내부의 동작에 주목하는 반면 [15-16, 5]는 페이지 요청 처리를 블랙박스로 보고 페이지 요청의 집합을 다룬다.

한편, 위 연구와 달리 페이지 요청 시 서버에서 일어나는 구체적인 동작을 분석하는 연구가 있다. 본 논문이 고려하는 이 주제의 기존 연구는 1장에서 서술하였다. 이와 같은 요청 처리 내부 동작 식별은 [1, 2, 4, 5, 8, 17]에서 언급하듯이 웹 앱을 재구조화하는데 도움을 줄 수 있다.

5. 결 론

웹 앱의 효율적인 검증이나 유지보수, 재구조화를 위해서는 페이지 요청 시 새 페이지 생성을 위해 일어나는 웹 컴포넌트 사이의 협력 과정에 대한 이해가 필요하다. 본 논문에서는 이러한 협력 과정을 자동적으로 추출하는 기법을 제안하였다. 제안하는 기법은 복잡한 웹 개발 기술에 덜 의존하는 동적 기법이다. 그래서 새로운 웹 개발 언어/기술이 도입되어도 제안하는 방법이 수정 없이 사용될 수 있다. 또한 웹 앱 소스를 블랙박스로 보고 서블릿 필터와 래퍼를 플러그인 하여 요청 처리 경로를 웹 앱의 기능 변화 없이 얻어낸다. 오픈 소스 웹

앱에 적용한 실험 통해 제안하는 기법의 유효성을 확인할 수 있었다.

향후 연구로는 정적 기법과의 결합을 통해 요청 처리 경로의 애매함을 줄이는 것이다. 둘째로, 여러 요청 처리 경로를 통합하여 웹 앱의 전체 아키텍처를 추출하는 것이다.

References

[1] M. Han and C. Hofmeister, "Modeling request routing in web applications," in *Proceedings of the WSE*, 2006.

[2] W. GJ. Halfond, "Identifying inter-component control flow in web applications," in *Proceedings of the ICWE*, 2015.

[3] G. C. Murphy et al., "Software reflexion models: bridging the gap between design and implementation," *IEEE Transactions on Software Engineering*, Vol.27, No.4, 2001.

[4] T. Parsons, A. Mos, and J. Murphy, "Non-intrusive end-to-end runtime path tracing for J2EE systems," *IEE Proceedings-Software*, Vol.153, No.4, 2006.

[5] M. H. Alalfi et al., "Automated reverse engineering of UML sequence diagrams for dynamic web applications," in *Proceedings of the ICSTW*, 2009.

[6] The Essentials of Filters [Internet], http://www.oracle.com/technetwork/java/filters-137243.html.

[7] D. Altar et al., "*Core J2EE Pattern,*" 2nd ed., Prentice Hall, 2003.

[8] J. Oh et al., "Automated transformation of template-based web applications into single-page applications," in *Proceedings of the IEEE COMPSAC*, 2013.

[9] JSR-000340 Java Servlet 3.1 Specification for Evaluation, http://download.oracle.com/otndocs/jcp/servlet-3_1-fr-eval-spec/index.html.

[10] A. Mesbah et al., "Crawling ajax-based web applications through dynamic analysis of user interface state changes," *ACM Transactions on the Web*, Vol.6, No.1, 2012.

[11] A. Marchetto et al., "Reajax: a reverse engineering tool for ajax web applications," *IET Software*, Vol.6, No.1, 2012.

[12] S. Mirshokraie et al., "Guided mutation testing for javascript web applications," *IEEE Transactions on Software Engineering*, Vol.41, No.5, 2015.

[13] S. Mirshokraie et al., "JSeft: automated javascript unit test generation," *in Proceedings of the ICST*, 2015.

[14] A. Guha et al., "Using static analysis for ajax intrusion detection," *in Proceedings of the WWW*, 2009.

[15] S. Hallé et al., "Eliminating navigation errors in web applications via model checking and runtime enforcement of navigation state machines," *in Proceedings of the ASE*, 2010.

[16] R. Rodriguez-Echeverria et al., "Generation of WebML hypertext models from legacy web applications," *in Proceedings of the WSE*, 2012.

[17] J. Oh et al., "MVC architecture driven restructuring to achieve client-side web page composition," *in Proceedings of the ICSESS*, 2016.

오 재 원

e-mail : jwoh@catholic.ac.kr

2004년 서울대학교 계산통계학과 (학사/석사/박사)

2004년~2007년 삼성전자 책임연구원

2007년~현 재 가톨릭대학교 컴퓨터정보공학부 부교수

관심분야 : Software Engineering, Web Engineering, System Software

안 우 현

e-mail : whahn@kw.ac.kr

1996년 경북대학교 전자공학과(학사)

1998년 KAIST 전기 및 전자공학과(석사)

2003년 KAIST 전자전산학과(박사)

2003년~2005년 삼성전자 책임연구원

2006년~현 재 광운대학교 컴퓨터소프트웨어학과 교수

관심분야 : 운영체제, 임베디드시스템, 시스템 보안

김 태 공

e-mail : sun@inje.ac.kr

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